TIETO- JA SÄHKÖTEKNIIKAN TIEDEKUNTA

Sampo Niittyviita

Testitapausten testikattavuuden riittämättömyys

Diplomityö
Tietotekniikan tutkinto-ohjelma
Marraskuu 2019

Niittyviita S. (2019) Testitapausten testikattavuuden riittämättömyys. Oulun yliopisto, tietotekniikan tutkinto-ohjelma. Diplomityö, 86 s.

TIIVISTELMÄ

Testitapausten luonti on keskeisessä roolissa ohjelmistotestauksessa. Tämä pätee niin testauksen toteutukselle, tutkimukselle kuin myös muulle alan teoreettiselle pohjalle. Testaus on hyvin pitkälti rakentunut testitapausten ympärille. Aiempi tutkimus on havainnut testitapauksilla testattaessa myös löytyvän sellaisia ohjelmistovirheitä, jotka eivät ole testitapausten määräämällä ohjelman alueella. Testitapauksia käytettäessä testauksessa, ne määräävät testattavasta ohjelmasta toiminnallisuuksien ja ominaisuuksien alueen, jota testit koskevat. Silloin on myös oltava ohjelman alue, joita testitapaukset eivät koske, ts., testitapausten ulkopuolinen alue. Testitapausten käsikirjoituksen määräämä testikattavuus ei siis välttämättä täysin vastaa toteutunutta testauksen laajuutta, kun testitapausten joukon testit ajetaan manuaalisesti.

Tämän opinnäytetyön tarkoituksena on tutkia tätä testitapausten testikattavuuden ulkopuolisten virheiden ilmiötä. Näin kaksi tutkimuskysymystä on esitetty: ”*Kuinka suuri osa käsikirjoitetun testauksen löytämistä ohjelmistovirheistä ovat käsikirjoitettujen testitapausten kattamia?” sekä* ”*Vaikuttaako testitapausten testausohjeistuksen rakenne ulkopuolisten virheiden esiintymiseen?”* Näihin kysymyksiin vastaamiseksi järjestettiin aikarajoitettu opiskelijoilla tehtävä koe manuaalisesta funktionaalisesti käsikirjoitetusta testauksesta, joka pohjautui ennalta luotuihin testitapauksiin.

Testikattavuuden yksiköitä usein luonnehditaan riittämättömäksi täysin kattavalle testaukselle, mutta tämän kokeen tulokset tuo tarkastelun alle vielä toisenlaisen ongelman testikattavuudesta: vaikka testitapausten määräämä testikattavuus olisi riittävä, se ei kuitenkaan välttämättä realisoidu riittävinä testauksen tuloksina. Vain osa virheistä löytyivät, jotka olisi pitänyt testitapauksia testaamalla löytää. Tämän lisäksi merkittävä osuus löytyneistä virheistä eivät olleet testitapausten kattamalla alueella.

Testitapausten ulkopuolinen olisi tiedostettava ja huomioitava paremmin. Sen huomiotta jättäminen ja testitapauksiin sokea luottaminen saattaa aiheuttaa ongelmia ja ylimääräisiä kustannuksia ohjelmistoalalla.

Avainsanat: ohjelmistotestaus, käsikirjoitettu testaus, ohjelmistovirhe, testitapausten joukko, manuaalinen testaus, funktionaalinen testaus, testitapauksiin pohjautuva testaus

Niittyviita S. (2019) The Insufficiency of Test Case Test Coverage. University of Oulu, Degree Programme in Computer Science and Engineering. Master's Thesis, 86 p.

ABSTRACT

The creation of test cases is in a central role in software testing. This applies to software testing in the software industry as well as its' research and other theoretical basis. Testing is largely based on test cases. Prior research conducted on test case-based testing has noted that the testing process discovers defects from the tested software which are not covered by the test cases' script, i.e., defects external to the test cases' coverage. When test cases are used in testing, they determine an area from the functionalities and other features of the software which the tests cover. This also means that there should exist an area outside this coverage: the external area of test coverage.

The purpose of this thesis is to study the phenomenon of the discovery of external defects of test case test coverage. Thus, two research questions have been formed as follows: *"How large part of the defects found by scripted testing are covered by the test coverage of the script?" and "Does the structure of the test script affect the emergence of external defects of the test case test coverage?"* To answer these questions an experiment of scripted testing was conducted with students as participants.

While the metrics of test coverage are often described as insufficient for complete testing, yet, as a result of this thesis, another issue of test coverage is brought upon inspection: even if the test coverage was sufficient; the testing conducted to achieve the coverage does not necessarily actualize as sufficient. Only a part of the defects were found with the tests which cover the defects and, in addition to this, a portion of the defects found by testing the test script are not actually covered by the script.

The external of the test case should be acknowledged and noted in greater detail. The ignorance of the external and the blind reliance on a test script may cause troubles and extra costs in the field of software testing.

Keywords: test suite effectiveness, test suite coverage, software, software testing, scripted testing, defect, bug, manual testing, functional testing, test case-based testing

SISÄLYSLUETTELO

ALKULAUSE

Olen suuresti kiitollinen mahdollisuudesta kirjoittaa tämä lopputyö sekä järjestää kokeen osana ohjelmistotekniikan kurssia. Kiitokset kuuluvat prof. Juha Röningille, Teemu Tokolalle, Christian Wieserille sekä Pekka Pietikäiselle. Opinnäytetyön tekemisen prosessi oli selkeä heidän ohjeistuksessansa. Aina kun oli kysyttävää tai apua tarvitsi, sitä oli saatavilla. Teemu antoi erinomaista kirjoitusteknistä opastusta opinnäytetyön kirjoittamiseen. Kiitokset Jaakko Sakaranaholle kaikesta hänen antamastaan tuesta sekä Prove Expertise Oy:lle. Kiitokset myös Satu Tammiselle hänen antamastaan avusta tilastotieteellisissä asioissa. Erityiset kiitokset kaikille kokeeseen osallistujille.

Olen myös kiitollinen, että olen saanut tämän mahdollisuuden tutkia oman työni kannalta näin läheistä aihealueita, ja vieläpä opiskelijakokeella; koen olevani etuoikeutettu sen suhteen. Oppimiskokemuksena opinnäytetyön tekeminen oli erinomainen.

Oulu, 10.11.2019

Sampo Niittyviita

LYHENTEIDEN JA MERKKIEN SELITYKSET

ASQ	American Society for Quality
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
ISTQB	International Software Testing Qualifications Board
ISO	International Organization for Standardization
NIST	National Institute of Standards and Technology
QAI	Quality Assurance International

1. JOHDANTO

Ohjelmistotestauksesta on tullut entistä merkittävämpi osa ohjelmistokehitystä, ohjelmistokehityksen lisääntyneiden laatuvaatimusten sekä niiden merkityksen kasvettua osana tätä kokonaisuutta [1, 2]. Kilpailu on ankaraa ohjelmistoalalla ja tämän kaltaisessa ympäristössä alhaisen laadun tuottaminen on erityisen vahingollista. Tässä ympäristössä ohjelmistotestaus on hyvä kilpailuvaltti, vaikkakin ohjelmistotestausta on luonnehdittu resurssi-intensiiviseksi sekä kalliiksi. Ohjelmistovirheiden löytäminen ja korjaaminen kuluttavat arviolta 30-50% ohjelmistoprojektien kokonaisajasta [4, 5]. Vuonna 2017 julkaistu kyselytutkimus, joka käsitteli ohjelmistotestauksen käytännön toteutuksen tilaa, esitti että 48%:ssa ohjelmistoprojekteista ohjelmistotestaus päättyy testauksen aikarajoituksiin tai testausbudjetin loppumiseen, eikä siis riittävän testauksen tason saavuttamisen seurauksena [6]. Tämä on testauksen tila, samalla kun ohjelmistovirheistä aiheutuneet kulut voivat olla huomattavan suuria: vuonna 2002 julkaistiin tutkimus National Institute of Standards and Techonology:n (NIST) toimesta, joka arvioi, että vuosittaiset kustannukset riittämättömästä testauksesta olivat 22,2 ja 59,5 miljardin dollarin välillä [7]. Ohjelmistotestauksen ja sen epäonnistumisen kustannusten seurauksena on ehdottoman tärkeää, että ohjelmistotestaus toteutetaan niin tehokkaasti kuin mahdollista ja parhain mahdollisin käytännöin.

Ohjelmistokehityksen voi toteuttaa monella eri tavalla sen päämäärien saavuttamiseksi. Osana ohjelmistokehitystä, ohjelmistotestausta voidaan myös toteuttaa monella eri tavalla: ohjelmistotestaajan saatavissa on monenlaisia menetelmiä, tekniikoita ja työkaluja ohjelmistotestausta varten. Sen lisäksi, että testaukseen on paljon lähestymistapoja, niin on myös tiedon lähteitä, ml.: testauskirjallisuus, tieteelliset julkaisut, opetus, koulutus ja standardit. Näin ollen, on testaajalla sekä hänen organisaatiollansa vaikea valinta tehtävänään: mitkä näistä metodeista, tekniikoista ja työkaluista olisi valittava testausta varten ja mistä sen tiedon tulisi ottaa?

Testitapauksia pidetään ohjelmistotestauksen toteuttamisen perustana ja testitapausten luontia tärkeimpänä testauksen osa-alueena [8, s. x; 9; 10]. IEEE määritelmän mukaan testitapaus on dokumentoitu spesifikaatio testauksen toteutukselle, testauksen tiloille ja ennustetuille tuloksille [12]. Juha Itkosen, Mika Mäntylän ja Casper Lasseniuksen vuoden 2007 tutkimuksessa käsikirjoitetun testauksen (engl. scripted testing) suorituskyvystä vain 82% löytyneistä ohjelmistovirheistä löytyivät suoraan käsikirjoitetuilla testitapauksilla [11]. Mikäli tämä sama ilmiö esiintyy laajemmin, niin se tarkoittaisi manuaalitestaajan toimesta tehtävän testauksen löytävän ohjelmistovirheitä myös ennalta määrätyn käsikirjoituksen testikattavuuden ulkopuolelta. Testauskäsikirjoituksen näin epäonnistuessa, on perusteltua väittää, että kehittämispotentiaalia testauskäsikirjoitusten laatimiseksi on olemassa. Mikäli testauskäsikirjoituksen testitapausten määräämän testikattavuuden ulkopuolella oleville ohjelmistovirheille voitaisiin määrittää yhteisiä ominaispiirteitä, niin testitapausten suunnittelua voitaisiin mahdollisesti parantaa näiden kautta. Lisäksi, niiden olosuhteiden tunteminen, joissa ohjelmistovirheitä löytyy käsikirjoitetulla testauksella testitapausten ulkopuolelta, voisi antaa selkeämmän kokonaiskuvan käsikirjoitetun testauksen toteutuksesta. Tunnistettaessa nämä mahdollisuudet, ohjelmistotestauksen testitapausten ulkopuolelta löytyviä ohjelmistovirheitä tutkitaan kahdella tutkimuskysymyksellä:

1. ”*Kuinka suuri osa käsikirjoitetun testauksen löytämistä ohjelmistovirheistä ovat käsikirjoitettujen testitapausten kattamia?”*
2. ”*Vaikuttaako testitapausten testausohjeistuksen rakenne ulkopuolisten virheiden esiintymiseen?”*

Näihin kysymyksiin haettiin vastausta *opiskelijoilla tehtävällä aikarajoitetulla kokeella manuaalisesta funktionaalisesta käsikirjoitetusta testauksesta, joka pohjautui ennalta luotuihin testitapauksiin.* Kokeesta kerätyn aineiston tueksi tutkimuskysymyksiin vastauksen saamiseksi luotiin uusi tapa tarkastella testitapausten ja virheiden suhdetta: ulkopuolisuuden tasot.

2. OHJELMISTOTESTAUS

Ohjelmistotestaus ohjelmistokehityksen osa-alueena on jatkuvan kehityspaineen alla oleva samoin kuin muukin ohjelmistokehitys. Ohjelmistoja käytetään jo osana lähes kaikkea ihmisen toimintaa. Siten varmasti jo kaikki ovat törmänneet ohjelmiin, jotka toimivat huonosti: laadukkaan ohjelman tuottaminen on haasteellista. Näin ovat todenneet myös monet ohjelmistotestauksen parissa toimivat [13]. Tämän vuoksi ohjelmistoteollisuudella on voimakas kannustin kehittää ohjelmistotestausta.

Vuonna 2007 julkaistu Antonia Bertolinon artikkeli ohjelmistotestauksen tilasta ja tavoitteista esitti neljä päämäärää ohjelmistotestauksen tutkimukselle:

1. universaali testauksen teoria,
2. testi-pohjainen mallintaminen,
3. 100 % testauksen automatisointi ja
4. maksimoitu testauksen tehokkuus [14].

Näihin tavoitteisiin pääseminen voisi edesauttaa ohjelmistotestausta merkittävästi, mutta näiden saavuttamiseksi on vielä matkaa jäljellä. Testauksen standardoinnista ja sertifioinnista huolimatta testauksella on hyvin vaihtelevat perustat ja toteuttamistavat, mikä näkyy selkeästi ohjelmistoalalle järjestetyissä kyselytutkimuksissa.

2.1. Ohjelmistotestaus osana ohjelmistokehitystä

Ohjelmistokehityksessä on käytössä useita eri ohjelmistokehitys-menetelmiä, kuten: vesiputousmalli, spiraalimalli ja erilaiset ketterät menetelmät. Vuonna 2017 tehdyn kyselytutkimuksen mukaan juuri ketterät menetelmät (esimerkiksi Scrum, Extreme Programming tai Feature-driven development) käsittävät noin 50 % ohjelmistoprojekteista [6]. Näistä suosituin ketterä menetelmä oli Scrum, jota käytti noin 80 % Agile-menetelmiä käyttäneistä. Muutaman vuoden takaisen (2013) kyselytutkimuksen mukaan pienemmissä yrityksissä käytetään enemmän Agile-menetelmiä perinteisiin verrattuna [15]. Tähän voi vaikuttaa suurten yritysten heikompi kykyä tehdä suuria organisaatiomuutoksia ja lisäksi pienet yritykset oletettavasti ovat keskimäärin nuorempia ja siten ovat jo perustamisvaiheissa ottaneet käyttöön uudempia ohjelmistokehitysmenetelmiä. Agile-menetelmistä esimerkiksi Scrummien Scrummit ovat osoittautuneet haasteellisiksi suurien organisaatioiden käytössä [16]. Scrummien Scrummien tutkimuksen mukaan erityisesti koordinaatio oli ongelma suurilla organisaatioilla [16]. Tämä saattaa myös olla ongelma ohjelmistotestauksessa.

Ohjelmistokehityksen mallilla on usein vaikutus siihen, kuinka ohjelmistotestaus toteutetaan. Agile-menetelmät tyypillisesti suosivat mm. automaatiota, testien kirjoittamista ennen koodia ja lyhyttä ja iteratiivista testausprosessia [17, 18, 19]. Perinteisemmissä menetelmissä, kuten esim. vesiputousmallissa testaus kohdennetaan ensisijaisesti valmiille tai lähes valmiille tuotteelle [18, 19].

2.2. Ohjelmistotestauksen toimiala

Ohjelmistotestauksen käytännöt ovat kokeneet muutosta vuosien varrella. Aikaisemmin ohjelmistotestauksen vastuu oli suurimmaksi osaksi ohjelmistokehittäjillä, kun viime aikoina ohjelmistotestauksen päävastuu on siirtynyt kehittäjiltä päätoimisten testaajien harteille [20]. Ohjelmistotestauksen tarpeet eri ohjelmistoille ovat hyvin yksilöllisiä. Tämä vaikuttanee osaltaan myös testauskäytäntöjen laajaan kirjoon: yksi lähestymistapa testaukseen ei välttämättä ole tehokas lähestymistapa kaikille ohjelmistoille. Ohjelmistotestauksen harjoittamisen kirjo näkyy selvästi myös testauksen kartoitustutkimuksissa. Usean kartoitustutkimuksen mukaan testaustyökalujen käyttö on vähäistä [6, 21, 22]. Näin on myös automaatiotestauksen laita [6, 23, 24, 25]. Merkittävä osuus ohjelmistoyrityksistä ei käytä systemaattisia testausstrategioita; kolmen kyselytutkimuksen tulokset systemaattisten testausstrategioiden käyttämisestä ohjelmistoyrityksissä vaihtelevat 25 ja 39 % välillä [5, 20, 26].

Myös kokemuspohjaiset testaustavat ovat yleisesti käytössä osana yritysten testausta. Eräässä 2011 julkaistussa kyselytutkimuksessa 82 % organisaatioista käyttivät kokemuspohjaisia testausmenetelmiä (mm. tutkivaa testausta) [20].

2.3. Ohjelmistotestauksen luokittelutavat

Ohjelmistotestaukseen on olemassa monenlaisia luokitteluperusteita, mm. käytetyn menetelmän, lähestymistavan tai testauksen ajankohdan mukaan. Ei olisi yllättävää, vaikka testaajat itsekkään eivät olisi täysin perillä kaikesta alan termistöstä.

Keskeisin ohjelmistotestauksen kahtiajako on testauksen jakaminen toiminnalliseen testaukseen (black-box-testaus) sekä rakenteelliseen testaukseen (white-box-testaus) [8, s.16-17]. White-box-testauksessa ohjelmiston sisäinen rakenne on tunnettu, kun black-box-testauksessa testaaja ei tunne ohjelman sisäistä rakennetta, eli – nimensä mukaisesti – ohjelman sisusta on musta.

Molemmat lähestymistavat testaukseen nähdään komplementaarisina menetelminä sen sijaan, että yhdellä pystyisi sulkemaan pois toisen testauksen kokonaisuudesta. Tosin, ainakin joillakin testauksen osa-alueilla white-box- ja black-box-menetelmillä ei välttämättä kuitenkaan ole merkittävää eroa; Henard ym. tutkimuksen mukaan: testitapausten priorisoinnissa black-box- ja white-box-menetelmillä regressiotestauksen testitapausten priorisoinnissa ei olisi merkittävän suurta eroa niiden suorituskyvyssä [28].

Ohjelmistotestaukseen on myös olemassa erilaisia testaustekniikoita (engl. testing technique tai testing method). Garousin ja Mäntylän jaottelu merkittävimmistä testausmetodeista on seuraavanlainen:

- mallipohjaiset- (engl. model-based),
- hakupohjaiset- (engl. search-based),
- mutaatiopohjaiset- (engl. mutation),
- kombinaatiolliset- (engl. combinatorial),

- symbolinen toisto- (engl. symbolic execution) ja
- satunnaistestausmetodit (engl. random testing method) [29].

Garousi ja Mäntylä vertasivat näiden metodien sekundääritutkimusten lukumäärää Googlen hakutulosten määrään, ja havaitsivat, että näiden välillä ei ollut suurta yhteyttä [29]. Tässä näkyy ohjelmistoteollisuuden ja testauksen tutkimuksen osittain eriävät kiinnostuksen kohteet.

Parhaimman metodin valinta ei välttämättä myöskään ole helppoa, ainakin, jos käytetään perusteena eri testaustekniikoista tehtyä tutkimusta. Hamlet on todennut testaustekniikoiden vertaamisen olevan haasteellista seuraavista syistä:

- empiiristen kokeiden osallistujien sekä kokeessa käytettyjen ohjelmien suhteelliset eroavaisuudet,
- oleellisten tutkimusparametrien valinnan vaikeus [30].

Ohjelmistolla itsellään on oma vaikutuksensa testauksen suorittamiseen, mm. ohjelmiston tarkoitus ja ominaisuudet asettavat omat muuttujansa testaukseen. Ohjelmistojen testattavuus on merkittävä tekijä ohjelmistojen testauksessa. Kaikista laajimman tutkimuskartoituksen ohjelmien testattavuudesta tehnyt tutkimus syntetisoi ohjelmiston testattavuuden määritelmään kahden pääasian ympärille:

- kuinka helposti ohjelma on testattavissa tai
- kuinka helposti ohjelmasta on löydettävissä virheitä [31].

Toisaalta myös esimerkiksi turvallisuuskriittisen ohjelman (mm. sotilaskäyttö tai terveydenhuolto) testaaminen voi edellyttää erilaista lähestymistä testaukseen kuin ei-turvallisuuskriittisiin ohjelmiin.

Ohjelmistotestauksen voi myös jakaa sen vaiheiden mukaan:

- yksikkö- (engl. unit testing),
- integraatio- (engl. integration testing),
- järjestelmä- (engl. system testing),
- hyväksyntä- (engl. acceptance testing) sekä
- regressiotestaukseen (engl. regression testing) [29].

Erityisen suosittu tutkimuksen kohde näistä testauksen osa-alueista on ollut regressiotestaus, oletettavasti sen kalleuden takia [29, 32].

2.4. Manuaalisen ja automatisoidun testauksen suhde

Täysin automatisoitu testauksen tulevaisuus on ollut monen haaveena jo pidemmän aikaa, kuten Bertolino sen on aiemmin todennut [14]. Täysin automatisoitu testaus ei kuitenkaan näytä ainakaan vielä olevan aivan välittömässä tulevaisuudessa, sillä kyselytutkimuksissa tehdyt arviot automaatiotestauksen suhteellisesta osuudesta testauksen kokonaisuudesta ovat alhaiset (arviot automaatiotestauksen osuudesta vaihtelevat 20 – 30 % välillä) [6, 20, 33]. Automaatiotestauksen osuuden ollessa näin alhainen sen voisi nähdä olevan useimmissa yrityksissä manuaalitestausta tukevana toimenpiteenä. Ongelmina automaatiotestaukselle on nähty mm.

- automaation ylläpidon vaikeus ohjelmiston muuttuessa nopeasti,
- automaatio-osaajien puute,
- työkalujen puute sekä
- haasteet testi datan ja testiympäristön saatavuuden kanssa [33].

Näin vuonna 2019 automaatiotestauksella ja manuaalitestauksella on omat roolinsa testauksen kokonaisuudessa, jotka eivät ainakaan vielä ole toistensa poissulkevia. Toisaalta myös automaatiotestauksen voisi nähdä jossain tapauksissa pitävän sisällään samoja testauksen prosesseja kuin manuaalitestauksenkin, jos ja kun automaatiotestien laatija on vuorovaikutuksessa testattavan ohjelman ja/tai sen testitapausten kanssa samoin kuin manuaalitestaajakin. Näin manuaalitestauksen ja automaatiotestauksen tutkimuksen havainnot voivat olla jossain määrin toisiaan tukevia.

Manuaalitestauksen tutkimuksella on oma paikkansa osana ohjelmistotestauksen tutkimusta, mutta sillä on haasteensa. Tämän opinnäytetyön yhteydessä tehdyn taustatutkimuksen perusteella ei löytynyt testauksen tutkimusta, joka olisi keskittynyt manuaalitestauksen testausaktiviteettiin. Siispä manuaalitestaus on edelleen relevantti tutkimuksen alue. Kuitenkaan yhtään sekundäärilähdettä manuaalitestauksesta ei ollut tehty vuoteen 2016 mennessä Garousin ja Mäntylän tekemän ohjelmistotestauksen tertiääritutkimuksen mukaan [29]. Myöskään tämän opinnäytetyönyhteydessä tehdyssä kartoituksessa ei löydetty vuonna 2016 tai sen jälkeen tehtyä manuaalitestauksen sekundäärtutkimusta.

2.5. Testauksen akkreditoinnin rooli ohjelmistotestauksessa

Testauksen toteuttamisen tueksi on olemassa organisaatioita, joiden tarkoituksena on pitää yllä ja levittää vakiintuneita ja hyviksi koettuja testauskäytäntöjä. Tällaisia organisaatioita ovat International Software Testing Qualifications Board (ISTQB), American Society for Quality (ASQ) ja Quality Assurance International (QAI). Testaussertifioinnilla on oma vaikutuksensa ohjelmistotestauksen alaan monien testaajien suorittaessa testaussertifikaatteja: pelkästään ISTQB-testaussertifikaatteja on myönnetty 641000 vuoden 2018 joulukuuhun mennessä [34]. ASQ ja QAI ovat myöntäneet huomattavasti vähemmän sertifiointeja testaukseen. Niiden yhteenlaskettu myönnettyjen sertifikaattien lukumäärä on noin 700000 vuoteen 2018 mennessä, mutta tähän on mukaanluettu myös lukuisia muitakin sertifikaattityyppejä kuin vain

ohjelmistotestaussertifikaatteja [35, 36]. ISTQB:n taas on erikoistunut ainoastaan ohjelmistotestauksen sertifiointiin.

2.5.1. Testausstandardit

Testausstandardien tarkoitus on käytännönläheinen niiden pyrkiessä parantamaan testauskäytäntöjä. Standardit ovat julkaistuja dokumentteja, jotka asettavat spesifikaatiot ja käytännöt maksimoidakseen luotettavuuden materiaaleille, tuotteille, käytänteille ja/tai palveluille, joita ihmiset käyttävät päivittäin [27].

ISO/IEC/IEEE 29119 -standardi on kansainvälisesti hyväksytty standardien joukko. Se on joukko ohjelmistotestauksen standardeja, joita voi käyttää mikä tahansa organisaatio tehdessään testausta missään muodossa [27].

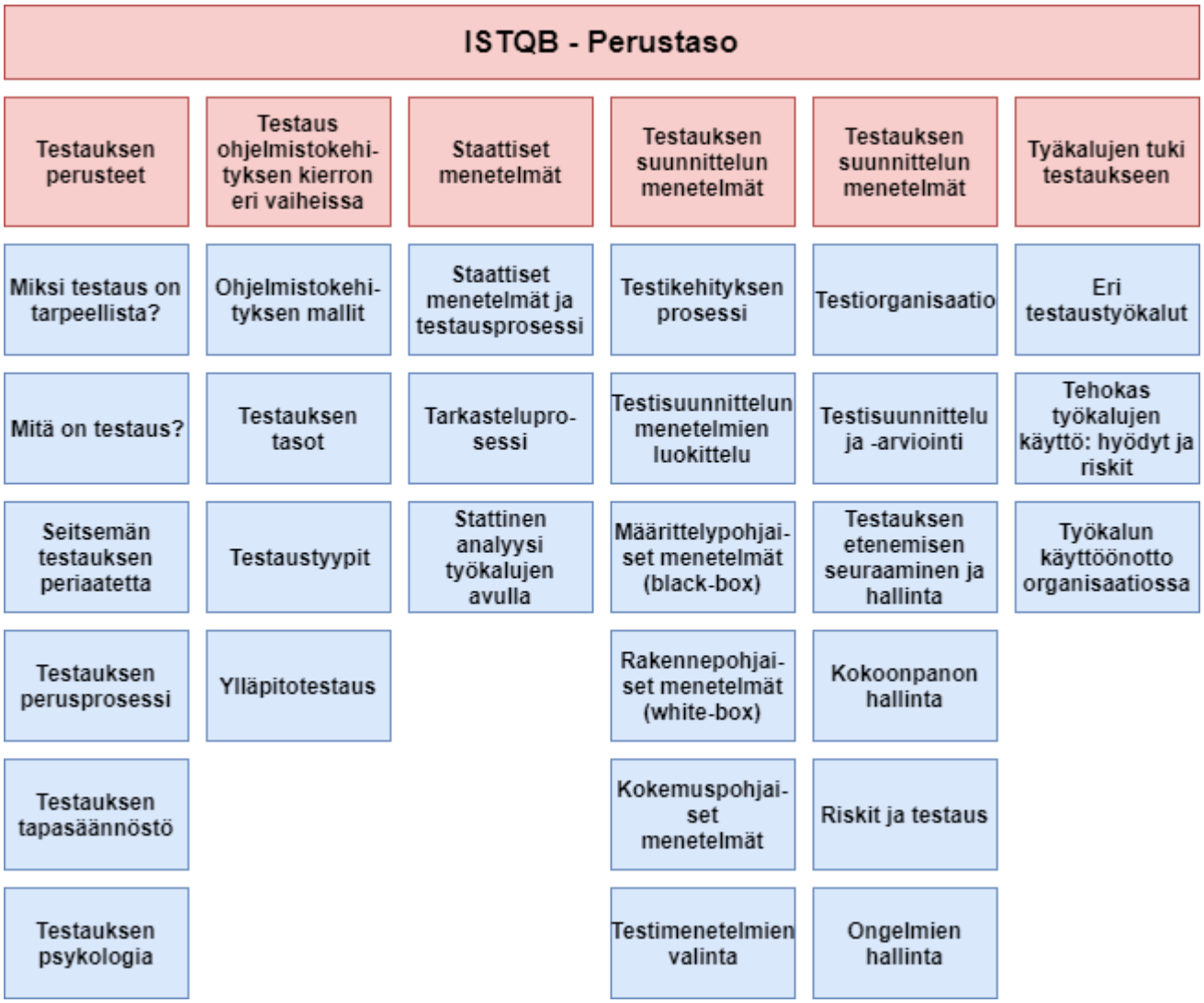
2.5.2. Testauksen sertifiointi

Testauksen sertifioimista tekevät mm. International Software Testing Qualifications Board (ISTQB), American Society for Quality (ASQ) ja Quality Assurance International (QAI). ASQ ja QAI ovat myöntäneet huomattavasti vähemmän sertifiointeja testaukseen. Niiden yhteenlaskettu myönnettyjen sertifikaattien lukumäärä on noin 700000 vuoteen 2018 mennessä, mutta tähän on mukaanluettu myös muitakin sertifiointeja kuin vain ohjelmistotestaussertifiointeja [35, 36]. ISTQB on näitä kahta organisaatiota huomattavasti nuorempi sen ollessa perustettu vuonna 2002. Se on myös myöntänyt huomattavasti enemmän sertifikaatteja ohjelmistotestauksesta: 641000 myönnettyä sertifikaattia vuoden 2018 joulukuuhun mennessä [34]. Monet toimijat pitävät ISTQB-sertifikaattia testauksen ohjelmistokehityksen tilan kuvana, ainakin Garousi V. ja Mäntylä M. ohjelmistotestauksen teritääritutkimuksessa näin toteavat [29]. Vähintäänkin ISTQB on organisaationa vaikutusvaltainen sen myöntämien sertifikaattien lukumäärän perusteella.

ISTQB on testaajia sertifioiva organisaatio, jonka tuntee suuri osa testaajista. Tähän tulokseen ainakin päädyttiin vuonna 2011 tehdyssä kyselytutkimuksessa, johon osallistuneista 89,6 % tunsivat ISTQB-koulutusjärjestelmän ja 67 % niistä, jotka tunsivat sen, olivat myös sertifioitu ISTQB-sertifikaatin perustasolla [20]. Tämä tulos on varmasti ainakin suuntaa antava. Vuoteen 2017 mennessä ISTQB-sertifikaatin tuntijoiden osuuden suuruusluokka testaajissa tuskin on muuttunut merkittävästi. Puhuttaessa näinkin laajasta sertifikaatin tuntemuksen tasosta: täytyy jo sertifikaatin oppisisällöllä olla havaittavia vaikutuksia yleisiin testauskäytäntöihin.

Kuva 1 on havainnollistus ISTQB-perustason oppisisällöstä [37]. Sertifikaatin saamiseksi, voi materiaalin käydä joko itse lävitse tai käydä koulutuksessa, jonka sisällöstä järjestetään sertifiointikokeita. ISTQB-perustason koulutukset ovat kestoltaan tyypillisesti 3 päivän mittaisia [38]. ISTQB-perustason oppisisältö on osa suurempaa ISTQB-sertifioidun testauksen sisällöllistä kokonaisuutta. Vuonna 2018 saatavilla olevia oppisisältöjä ISTQB:n tarjoamana oli kaiken kaikkiaan 10 kappaletta,

johon kuuluu mm. testauksen hallintaa, Agile-testausta, testausautomaatiota ja mallipohjaista testausta [39].



Kuva 1. ISTQB - perustason oppisisällön yhteenveto.

3. KÄSIKIRJOITETTU TESTAUS

Ohjelmistotestauksen yhtenä keskeisimmistä osista pidetään testitapausten laatimista. Määritelmältänsä testitapaus on testauksen pienin tarkasteluyksikkö, eikä sitä siten enää voida jakaa pienempiin testauksen osiin [12, s. 466]. Käytettäessä näiden testitapausten kirjoitettuja testausohjeita testauksen perustana, on tällöin kyseessä käsikirjoitettu testaus [12, s. 403]. Testitapaukset ovat merkittävässä roolissa testauskirjallisuudessa sekä testauksen tutkimuksessa [8, 9, 10]. Testitapauksia käytetään – ei pelkästään testauksen perusyksikkönä – vaan myös testauksen metriikkana testauksen tilan ja sen etenemisen havainnoimiseksi. Testauksen tilojen määreistä on johdettavissa erilaisia testikattavuuden yksiköitä (kutsutaan myös metriikoiksi) [12, s. 467].

3.1. Testitapausten perusteet

ISO/IEC/IEEE 24765:2017 määritelmän mukaan testitapaus on dokumentoitu spesifikaatio testauksen lähtötilanteelle, toteutukselle, toimenpiteille ja odotetuille tuloksille, joilla varmistetaan testauksen tavoitteisiin pääsy: oikea ohjelman implementaatio, virheiden tunnistaminen, laadun ja muun arvokkaaksi koetun tiedon hankkiminen [12, s. 466]. Testitapaus voi olla joko automatisoitu, jolloin voidaan käyttää ilmaisua testiohjelma (engl. test script), tai manuaalinen eli ihmisen testattava [12, s. 466, 472]. Konkreettinen esimerkki testitapauksen rakenteesta on esitettynä kuvassa 2, joka on Microsoftin käyttämä testitapauspohja [40, s. 210]. Kyseisen kuvan testitapauspohjasta käy ilmi testattavat ohjelman alueet, testin tarkoitus ja tärkeys, siihen vaadittu aika sekä varsinaisen testin lähtökohdat, tehtävät toimenpiteet ja odotetut tulokset. Mikäli suorittaa Googlen kuvahaun hakusanoilla: ”test case template”, on löydettävissä lukuisia samaa kaavaa noudattavia testitapauspohjia pienin variaatioin.

Testitapauksen nro.: 1			
Alue: Selkeä ja lyhyt testitapauksen otsikko			
Alue:		**Alueen osa:**	
Prioriteetti	**Tyyppi**	**Testin toistaminen**	**Testauksen aika (minuuttia)**
1	Toiminnallisuus	Jokainen versio	2
Kuvaus:			
Testin tarkoitus: Lähtötilanne ja tausta: Toimenpiteet: 1. 2. 3. Odotetut tulokset: Muistiinpanot:			

Kuva 2. Esimerkki Microsoftin käyttämästä testitapauksen rakenteesta.

3.2. Testitapausten laatiminen

Testitapausten laatimiseksi on olemassa erilaisia tekniikoita. Moni testauksen tutkija pitää (mm. Garousi ja Mäntylä [29]) ISTQB-sertifikaatin esittämää kuvaa testauksen käytännöistä kuvaavana testauksen harjoittamisen tilasta (ISTQB-sertifiointi tarkemmin luvussa: 2.5.2. Testauksen sertifiointi). Se on myös hyvin laajalle levinnyt: ISTQB on myöntänyt 641000 sertifikaattia vuoden 2018 joulukuuhun mennessä [34]. ISTQB perustason oppisisältö jaottelee testitapausten laatimisen kolmelle menetelmälle:

- black-box (toiminnallinen),
- white-box (rakenteellinen) sekä
- kokemuspohjainen.

Kokemuspohjaisessa testitapausten suunnittelussa testitapaukset johdetaan testaajan, kehittäjän, käyttäjän tai muun asianosaisen tietämyksestä ja kokemuksesta. White-box-testitapausten (rakenteellisten testitapausten) suunnittelussa testitapaukset johdetaan ohjelmiston arkkitehtuurista, yksityiskohtaisesta mallista tai muusta ohjelmiston rakenteesta kertovasta informaatiosta. Black-box-testitapausten

(toiminnallisten testitapausten) suunnittelussa testitapaukset johdetaan ohjelmiston vaatimuksista, määritelmistä, käyttötapauksista tai -kertomuksista [37, s. 57].

Testitapauksissa tehtäviä virheitä ovat mm. puuttuvat stepit, liiallinen runsassanaisuus, vaikean ammattikielen käyttö (esim. koodiin viittaaminen tai lyhenteiden käyttö), jota testaaja ei välttämättä ymmärrä ja epäselvät läpäisy- tai epäonnistumiskriteerit [40, s. 213]. Laadittaessa testitapauksia, on myös hyvä ottaa huomioon mahdolliset negatiiviset testitapaukset (kuinka ohjelman ei tulisi toimia) positiivisten testitapausten lisäksi (kuinka ohjelman tulisi toimia).

3.3. Testitapausten soveltaminen

Vaikka testitapauksille on olemassa formaali rakenne, aivan kaikki eivät kuitenkaan käytä sellaista. Vaihtoehtoisesti testitapauksia kuvataan epäformaalimmin kuvauksina sekä usein myös ilman odotettuja tuloksia [6]. Organisaation käyttäessä odotettuja tuloksia laadituille testitapauksille, varsinaisten tulosten ja odotettujen tulosten vertailua ei välttämättä kuitenkaan suoriteta [6]. Toisin sanoen, *yksi testitapauksen kirjoitusasu voi johtaa useampaan erilaiseen testausaktiviteettiin* riippuen organisaation toimintakulttuurista.

Testitapausten kirjoittaminen on usein nähty hyödyllisenä käytäntönä ennen koodin kirjoittamista, kuitenkaan se ei ole kovin yleinen käytäntö ohjelmistoyrityksissä: vuonna 2017 tehdyn kyselytutkimuksen osallistujien yrityksistä vain 26 % kirjoitti testitapauksia ennen koodin kirjoittamista [6]. Siitäkin huolimatta, että kyseisen tutkimukseen osallistujien yrityksistä 66 % suosi testitapauksen laatimista ennen koodin kirjoittamista [6]. Testaukselle asetetut toiveet ja tavoitteet eivät välttämättä ole realisoitavissa. Olisi varmasti hyödyllistä testata ohjelma aina tietyllä varmasti toimivalla kaavalla ja täysin kattavasti, mutta käytännössä tämä ei kuitenkaan ole mahdollista. Sen esteenä voivat olla mm. rajalliset testausresurssit, testauksen prioriteetit tai ammattitaidon puute.

Page, Johnston ja Rollison esittävät, että kaikkia testejä, kuten joitain erikoisempia testejä, ei välttämättä ole tarvetta testata kuin kerran tai kaksi, jolloin testitapauksen laatimiseen ja muuhun oheistoimintaan ei välttämättä ole järkevää käyttää aikaa [40, s. 211].

3.3.1. Inattentiivinen sokeus testitapauksiin

Mäntylä M. ja Itkonen J. käsittelevät artikkelissaan useamman testaajan vaikutusta testauksen tuloksiin. Tutkimuksen mukaan viiden henkilön käyttäessä kaksi tuntia yhden alueen testaamiseen, niin tällöin löytyy 71% enemmän virheitä kuin tilanteessa, jossa yksi henkilö käyttäisi saman aikamäärän (10h) saman ohjelman testaukseen [41]. Artikkelissaan he myös yhdistivät inanttentiivisen sokeuden ohjelmistotestaukseen, josta he mainitsevat esimerkkinä erään tunnetun nettisivun, jossa ihmisiä pyydetään laskemaan videolta; kuinka monta kertaa siinä esitetyssä koripallo-ottelussa heitetään palloa [41, 42, 43]. Tämän jälkeen videon katsoneilta heiltä kysytään: näkivätkö he gorillan videolla. Yli puolet ihmisistä eivät huomaa gorillaa videolla. Mäntylä ja Itkonen käsittelevät ilmiötä yhtenä selittävänä tekijänä eri testaajien testaustulosten

vaihtelevuudelle. Sitä voisi kuitenkin katsoa myös testitapausten näkökulmasta: jos testauksen fokus on keskittynyt testitapausten määräämälle alueelle, voi testitapausten ulkopuolinen osuus mahdollisesti jäädä ns. gorillaksi, ja vieläpä vaihdella riippuen testaajasta.

Voisi myös spekuloida, että osa testitapausten toistamisen tuottamista tuloksista syntyisi juurikin testitapausten ulkopuolisen alueen testaamisesta testitapauksia toistettaessa, varsinkin, jos sitä tekee useampi eri henkilö. Yksi käsikirjoitetuissa testitapauksissa nähdyistä hyödyistä on juuri toistettavuus, ja erityisesti regressiotestauksen yhteydessä jo olemassa olevia testejä toistetaan [40, s. 211].

3.4. Hyödyt ja haitat

Testitapausten käytölle on nähty sekä hyötyjä, että haittoja. Page A., Johnston K. ja Rollison B. tuovat esille testitapausten hyötyinä kirjassaan ”How we test at Microsoft”;

- testauksen etenemisen seurauksen,
- toistettavuuden sekä
- menneisyyteen viitattavuus [40, s. 211].

Kun testeille on olemassa hyvin dokumentoidut testitapaukset ne ovat helpommin toistettavissa tarpeen tullen esimerkiksi regressiotestauksen yhteydessä. Lisäksi testauksen etenemistä voi seurata erilaisten metriikoiden esim. testitapausten läpäisy prosentin avulla. Tehdyistä testitapauksista voi myös pitää yllä niiden testausajankohdat ja testatuista alueista tietoa. Testitapausten haittoina puolestaan Page A., Johnston K. ja Rollison B. nostavat esille:

- dokumentaation ajankäytön,
- testitapausten päivittämisen tarpeen sekä
- testitapauksia laadittaessa testitapauksen lukijan tietotason arvioimisen [40, s. 211].

Testitapauksen laatimiseen vaadittava aika voi olla suuri suhteessa itse testaukseen vaadittavaan aikaan, jolloin testitapauksen laatiminen ei välttämättä ole järkevää. Testitapauksen laatimiseen vaadittua aikaa on käsitellyt myös mm. Niittyviita [44]. Ohjelmistojen muuttuessa ja kehittyessä samalla myös testitapauksille syntyy paine sopeutua muutokseen eikä vanhat testitapaukset välttämättä enää ole päteviä, ja niitä on sitten päivitettävä ajan tasalle. Laadittaessa testitapauksia testitapauksiin on sisällytettävä tietoa, joka voi tuntua testitapauksen laatijalle päivän selvältä, ja siten se voi olla hankalaa tuoda esille testitapauksessa. Testitapauksen lukijalle tämä informaatio ei välttämättä kuitenkaan ole itsestään selvää. [40]

Selkeästi testitapauksille on tunnistettavissa hyötyjä kuin myös haittoja. Käytännössä testitapausten soveltaminen onkin usein vain osa testauksen kokonaisuutta, mikä käy konkreettisesti ilmi kyselytutkimuksista, joilla on kartoitettu testaukseen liittyviä käytäntöjä, mm. Kassab ym. (2017) ja Haberl ym. (2011) [6, 20].

4. TESTIKATTAVUUS

Haluttaessa mitata testauksen kattavuutta testattavasta ohjelmasta on tätä varten olemassa testikattavuuden mittayksiköitä. Testikattavuudella viitataan testitapausten määräämään osuuteen testattavan ohjelman vaatimuksista tai muista kattavuuden perusteina käytetyistä määreistä [12]. Testikattavuudesta on olemassa monenlaisia yksiköitä, joiden perustana voidaan käyttää mm. koodia, käyttöliittymän komponentteja tai tapahtumasarjoja [45]. Jo testikattavuuden perusteissa on omat haasteensa: ohjelman toiminta ei välttämättä asetu samalle abstraktion tasolle, josta se olisi testattavissa (luku 4.2. Testauksen kolmijako käsittelee tätä).

Yksi tapa tarkastella testauksen tavoitetta on riittävän testikattavuuden (engl. test coverage) saavuttaminen, kuten Ammann P. ja Offutt J. sen toteavat paljon viitatussa kirjassaan: ”Introduction to Software Testing” [10, s. xv]. Testikattavuutta usein kuitenkin luonnehditaan riittämättömäksi testauksen mittaamiseksi, mutta niitä kuitenkin käytetään laajalti testauksen sekä sen tutkimuksen yhteydessä [46]. Tämä riittämättömyys on ongelmallista sen käyttämisen vuoksi: virheelliset päätökset riittämättömien mittayksiköiden vuoksi voivat koitua kalliiksi – *testikattavuuden ulkopuolinen alue on siten tärkeä tiedostettava sekä tutkittava testauksen osa-alue*. Testikattavuuden ulkopuolisen alueen tarkastelu ei välttämättä myöskään ole rajattavissa pelkästään erilaisten testikattavuuden yksiköiden suhteen, vaan koko testausprosessin: manuaalisen käsikirjoitetun testauksen tuloksena vaikuttaisi myös löytyvän virheitä testitapausten ulkopuolelta [11].

4.1. Testikattavuuden yksiköt

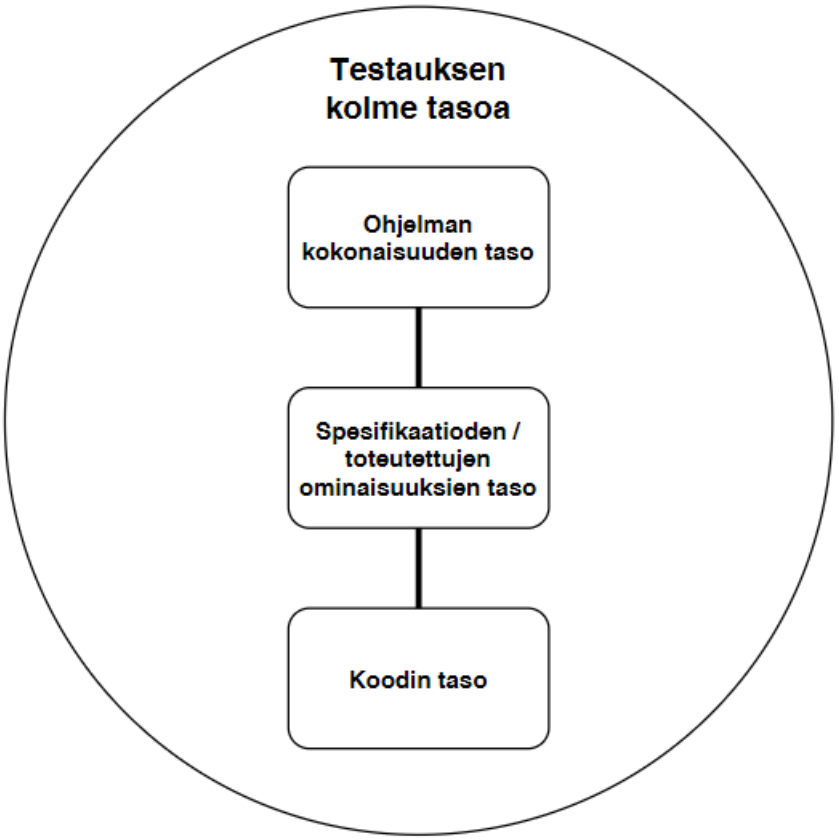
Testikattavuutta voidaan mitata lukuisilla erilaisilla yksiköillä. Vaatimuskattavuus on yksi suosittu metriikka testauksen kattavuudelle [6]. Vaatimusten lisäksi kattavuuden perusteina voidaan käyttää, mm. koodia, käyttöliittymän komponentteja tai tapahtumasarjoja [45]. Muita testikattavuuden metriikoita on mm. virheiden lukumäärä, virheitä löytävien testien lukumäärä, testitapausten lukumäärä, testien läpäisy- tai epäonnistumissuhde [40, s. 211]. Lause- ja haarakattavuus (engl. statement and branch coverage) ovat käytettyjä testikattavuuden yksiköitä, joiden toimivuudesta on olemassa tutkimusta [47, 48, 49]. Testikattavuuden yksiköiden tarkoituksena katsotaan olevan testauksen edistymisen mittaaminen sekä niitä voidaan käyttää informaation lähteinä päätettäessä koska testauksen voi lopettaa [10]. Erityisesti silloin kun ohjelmasta ei löydy enää virheitä, testikattavuuden yksiköt nähdään hyödyllisinä [46].

Virheiden ennustamista voidaan käyttää myös arvioitaessa testikattavuutta. Ostrand:in & Weyuker:in virheiden jakaumaa käsittelevän tutkimuksen mukaan (vuodelta 2002); valtaosa virheistä esiintyy pienessä osassa ohjelman koodia, muistuttaen Pareto-jakaumaa (noin 80 % virheistä löytyy noin 20 % ohjelman koodista) [50]. Lisäksi, samoista ohjelman alueista, joista on aiemmin löytynyt virheitä, löytyy vielä myöhemminkin virheitä [50]. Tämä ei kuitenkaan ole ainoa virheiden ennustamiseen käytetty perusta: muitakin erilaisia metriikoita on kehitetty lukuisia [51].

4.2. Testauksen kolmijako

Testikattavuuden määrittämisen perusteilla (mm. koodilla, käyttöliittymän komponenteilla tai tapahtumasarjoilla) on omat haasteensa käytettäessä niitä testikattavuuden määrittämiseen. Tapahtumasarjat, joita graafisen käyttöliittymän testauksessa käydään lävitse, ovat korkeammalla abstraktion tasolla kuin sen koodi, minkä vuoksi pelkkää koodia ei välttämättä yksistään voida käyttää riittävänä perusteena testauksen suorittamiseksi. Näin totesivat Memon, Soffa ja Pollack artikkelissaan käyttöliittymien kattavuutta käsittelevässä artikkelissaan [52]. Siitä syystä koodista yksistään ei välttämättä vielä voida johtaa riittävästi testejä graafisen käyttöliittymän tapahtumasarjojen testaamiseksi (testaushan ei ole rajoittunut pelkästään white-box-testaukseen).

Tarkasteltaessa samankaltaisesti testausta spesifikaatioiden näkökulmasta; vaikka testaamalla pelkästään spesifikaatioita voitaisiin todeta kaikkien spesifikaatioiden toteutuvan – kuitenkin spesifikaatiot muodostavat vielä korkeamman abstraktion tason spesifikaatioista: ne muodostavat oman kokonaisuutensa. Esimerkiksi, sama spesifikaatio on saatettu tulkita kahdessa eri paikassa hieman eri tavalla ja toisaalta, muodostavatko spesifikaatiot toisiinsa nähden järkevän kokonaisuuden tai ovatko spesifikaatiot ylipäätään järkeviä. Spesifikaatio voi itsessään olla hyvä ja sen pohjalta rakennettu ohjelman osa voi olla toimiva kokonaisuus, mutta näin ei välttämättä ole, kun puhutaan spesifikaatioiden joukosta.



Kuva 3. Testauksen kolmijako: koodin, spesifikaatioiden/ominaisuuksien ja ohjelman kokonaisuus.

Ohjelman koodi, spesifikaatiot tai toteutetut ominaisuudet (sillä spesifikaatioita saattaa puuttua) sekä ohjelman kokonaisuus voitaisiin näin esittää kolmella abstraktion tasolla, joista minkään yksittäisen abstraktion tason testaaminen ei vielä välttämättä riitä testaamaan muita tasoja riittävän hyvin. Kuva 3 havainnollistaa tätä testauksen tarkastelua kolmen abstraktion tason kautta. Välttämättä myöskään virhe yhdellä abstraktion tasolla ei realisoidu virheeksi toiselle tasolle, esimerkiksi, koodin tasolla voi olla virhe, joka ei näyttäydy millään konkreettisella tavalla käytettäessä ohjelmaa tai vastaavasti ohjelman käyttöliittymän tasolla voi olla virhe, joka ei ole itsessään virhe koodin tasolla.

4.3. Testikattavuuden ongelma

Testikattavuuden hyödyllisyyden mittayksikkönä kyseenalaistavaa tutkimusta löytyy ainakin jo 2000-luvun vaihteesta. Silloin Lionel Brand ja Dietmar Pfahl tutkivat: onko testikattavuuden käyttämisellä testitapausten luonnin perustana erillistä vaikutusta virhekattavuuteen, kun otetaan huomioon testausintensiteetti [53]. Tutkimus antoi osviittaa, että testikattavuuden yksikön suoma hyöty käytettäessä sitä testitapausten luonnin perustana ei itsessään aiheutuisi testikattavuuden, vaan ainoastaan sen tuoman lisätestauksen kautta, mikä tarkoittaisi, että samaan, tai lähes samaan testauksen lopputulokseen päästäisiin vain lisäämällä testausintensiteettiä. Myös mm. Staats, Gay, Whalen sekä Heimdahl ovat tutkineet kattavuuden vaikutusta testauksen tuloksiin. Haara- tai MC/DC-kattavuuden käyttäminen yksistään testien luomisen perustana on tehottomampi vaihtoehto pelkälle satunnaistestaukselle ilman mitään käytettyä testikattavuuden yksikköä [54, 55]. Näissä haara- ja MC/DC-kattavuuden tutkimuksissa kuitenkin testikattavuuden käyttäminen satunnaistestauksen tukena tuotti paremmat lopputulokset kuin pelkkä satunnaistestaus.

Testitapaukset nähdään tyypillisesti testauksen perusyksikkönä (ISO/IEC/IEEE 24765:2017) [12]. Testaus näin muotoiltuna – teoreettisesti täydellisesti kattavat testit edellyttäisivät tällöin täydellistä testitapausten joukkoa. Tämän olisi pidettävä paikkaansa, testikattavuuden ja testitapauksen ISO/IEC/IEEE 24765:2017 määritelmien pohjalta.

Testikattavuus voi kuitenkin jo määritelmällisesti olla harhaanjohtava, jos testausta tarkastellaan sen käytännön soveltamisen näkökulmasta. Kuten Page A., Johnston K. ja Rollison B. kirjassaan “How we test at Microsoft” mainitsevat: “… testitapaukset eivät määritä *kaikkia* testausaktiviteetteja.” [40, s. 212]. Muunlaistakin testausta kuin vain testitapauksiin perustuvaa harjoitetaan [20; 40, s. 212]. Testauksen toteutus organisaatiotasolla onkin harvoin eksklusiivisesti testitapauksiin perustuvaa [6]. Lisäksi, testitapauksen suoritus ei välttämättä realisoidu aina samalla tavalla: ajallisesti sama testaussuoritus eri henkilöiden tekemänä voi toteutua eri tavoin [41]. Konsensusta ei myöskään ole muodostunut koodikattavuuden ja testitapausten joukon suhteelle toisiinsa nähden, ts., minkälainen ja kuinka suuri vaikutus testitapausten kattavuudella on testauksen tuloksiin [56]. Otettaessa kaikki nämä huomioon ja tarkasteltaessa niitä suhteessa testitapausten testikattavuuden määritykseen jää tällöin väistämättä testausta sekä virheitä kattavuuden määritelmän ulkopuolelle. Siten *määritelmän mukaisen testikattavuuden saavuttaminen tuskin on vielä todella saavutettu testikattavuus ohjelman tasolla*, vaikka se teoriassa olisikin mahdollista.

Tällä on huomattavia potentiaalisia ongelmia. Jos testauksen tutkimusta tai testausta ylipäätään tarkastellaan pelkästään testitapausten näkökulmasta, ei testaus voi tuottaa riittäviä tuloksia. Testauksen tutkimuksen on syytä ottaa tarkempaan käsittelyyn testitapausten ulkopuoli.

Huolimatta testikattavuuden rajallisuudesta testauksen tilan tarkastelemiseksi testikattavuuden yksiköitä kuitenkin käytetään laajalti testauksen mittaamiseksi, vaikkakin tämä rajallisuus usein todetaankin alan kirjallisuudessa [46]. Useimmilla ohjelmilla voitaisiin testata lähes ääretön määrä erilaisia ohjelman tilojen kombinaatioita. Näin monen tilan testaaminen ei vain ole mahdollista, jolloin testien täydellinen kattavuus on käytännössä mahdotonta. Kuhn D., Wallace D. sekä Gallo A. käyttivät esimerkkiä laitteesta, jolle on 20 syötettä, joille kaikille on olemassa 10 mahdollista arvoa; tälle laitteelle on silloin olemassa 10^{20} eri asetusten kombinaatiota [3].

On siis perusteltua olettaa testikattavuudella olevan ulkopuolinen alue.

4.4. Testitapausten testikattavuuden ulkopuoli

Juha Itkosen lisensiaatintyössä vuodelta 2008, käsikirjoitetun testauksen (engl. scripted testing) löytämistä ohjelmistovirheitä vain 82% löytyivät suoraan käsikirjoitetuilla testitapauksilla [11]. Tätä tulosta ei kuitenkaan esitetty lisensiaatintyön pohjalta tehdyssä konferenssijulkaisussa. Tutkimusta, joka tutkisi juuri näitä käsikirjoituksen ulkopuolelta löytämiä virheitä ei löydetty tämän opinnäytetyön taustatutkimuksen yhteydessä. Mitä tulee IEEE:n tai ISTQB:n virheiden luokituksiin; ulkopuolisuutta käsittävää termiä tai yksikköä ei myöskään sieltä löydetty.

Testikattavuuden yksiköiden käyttämisen on todettu olevan riittämätön tapa mitata riittävän testauksen saavuttamista [46]. Tämän ongelman voisi pukea myös toisin: testien ulkopuolista ei tunneta riittävän hyvin. Jotta löytäisi testien ulkopuolisen; tuntemattoman, on tehtävä tutkimusmatka ohjelman tuntemattomaan – ja tässä, nimensä mukaisesti, tutkiva testaus voisi olla hyödyksi. Tutkivasta testauksesta ei ole tehty yhtään sekundääritutkimusta, kuin ei myöskään manuaalisesta testauksesta ylipäätään, kuten käy ilmi Garousin ja Mäntylän testauksen tertiääritutkimuksesta [29]. Tutkivasta testauksesta on vain vähän tutkimusta, mutta tutkimusta, mitä siitä on tehty vaikuttaa lupaavilta sen testauksen tehokkuuden ajankäytöllisesti sekä löytämien virheiden laadun puolesta [57, 58, 59, 60].

Tosin, ainakin osittain testausta harjoitetaan myös muutoinkin kuin vain pelkästään testitapausten avulla [20; 40, s. 212]. Monet yritykset ovat mahdollisesti ratkaisseet ulkopuolisen ongelman mukautumalla kehitettävän ohjelmansa tuomiin haasteisiin sopeuttamalla ohjelmistokehityksen ja testauksen tähän (esim. Agile-menetelmin) tai ammattitaitoisten testaajien avulla, jotka ovat kokemuksensa kautta oppineet testaamaan myös sitä, joka jää helposti testien ulkopuolelle tuntemattomaksi.

Itse ulkopuolisten virheiden esiintymiselle voi löytää useita eri selityksiä. Jos käytetyt testitapaukset ovat lähtökohtaisesti riittämättömät; saattaa ulkopuolisia virheitä esiintyä enemmän. Toisaalta, vaikka testitapaukset olisivat riittävät määriteltyyn testikattavuuteen, testikattavuuden riittämättömyys saattaa johtaa ulkopuolisten virheiden suurempaan esiintymiseen. Selitys saattaa löytyä myös

käytetystä lähestymisestä testaukseen: käsikirjoitettu testaus. On myös mahdollista, että testauksessa aina jossain määrin testaajan henkilökohtaiset ominaisuudet ja taidot vaikuttavat testauksen toteutukseen, mitkä johtavat uusien ohjelmistovirheiden löytymiseen (Itkonen tuo tämän myös esille) [58].

Ei välttämättä ole helppoa luoda testitapauksia yllättävälle ja epätavalliselle. Vastaavasti tämän vastakohtana, kuten Page, Johnston ja Rollison toteavat: testin kirjoittajan itsestään selvästä voi olla haasteellista laatia testitapauksia: ei välttämättä ole helppoa hahmottaa, mikä on muille selvää ja mikä ei [40, s. 211].

5. MENETELMÄ

Käsikirjoitettujen testitapausten noudattamisen mittaamiseksi suunniteltiin kahden tunnin kestoinen koe, johon osallistui tekniikan alojen opiskelijoita. Opiskelijat jaettiin kahteen eri ryhmään testaamaan heille laadittujen testitapausten avulla. Molemmilla ryhmillä oli testattavanaan kaksi ohjelmaa, joista kumpikin ryhmä testasi yhden ohjelman ns. kevyillä ja toisen raskailla testitapauksilla. Tämän asetelman tarkoituksena oli mitata testitapausten vaikutusta testauksen tuloksiin. Lisäksi, testaajien taustaa kartoitettiin erityisesti testaukseen ja ohjelmointiin liittyen.

Jotta kokeen järjestelyt olisivat mahdollisimman toistettavissa ja selkeä; kokeen suunnittelussa tukeuduttiin IEEE:n testausstandardeihin. Aikaisemmassa luvussa 2.7. Testauksen akkreditoinnin rooli ohjelmistotestauksessa on käsiteltynä standardit tarkemmin. Kokeen testitapausten suunnitteluvaiheessa ohjelmat annettiin testattavaksi kahdelle testauksen ammattilaiselle testattavien ohjelmien virheiden kartoittamiseksi. Koeasetelma testattiin kokonaisuudessaan kahdella vanhemmalla opiskelijalla ennen kokeen varsinaista järjestämistä mahdollisten virheiden ja puutteiden korjaamiseksi koeasetelmasta sekä kokeen testitapausten testauksen ajankäytön tarkemmaksi arvioimiseksi.

Kaikista varotoimenpiteistä huolimatta jo suunnitteluvaiheessa oli ilmeistä, että kaikkiin tämän kaltaisen kokeen vaikuttavia tekijöitä ei pystytä täysin kontrolloimaan, ainakaan tämän laajuisella kokeella. Harkituilla koeasetelman vaihtoehdoilla on kullakin omat rajoituksensa.

5.1. Tutkimuskysymykset

Ulkopuolisten ohjelmistovirheiden tutkimiseksi esitettiin kaksi tutkimuskysymystä, joista ensimmäinen on muotoa: ”*Kuinka suuri osa käsikirjoitetun testauksen löytämistä ohjelmistovirheistä ovat käsikirjoitettujen testitapausten kattamia?”* Vastaus tähän kysymykseen saadaan tarkastelemalla: onko löytynyt ohjelmistovirhe löydettävissä testitapausten eksplisiittisillä ohjeilla? Mikäli löytyy; on ohjelmistovirhe testitapausten sisäpuolella, ja jos taas ei löydy: on ohjelmistovirhe testitapausten ulkopuolella. Näin saadaan määritettyä suhdeluku testitapausten sisäpuolisille sekä ulkopuolisille ohjelmistovirheille. Hypoteesit ensimmäiselle tutkimuskysymyksille ovat:

H_0: käytettäessä testitapausten testausohjeita testattaessa ohjelmistoa, kaikki löytyneet ohjelmistovirheet ovat löydettävissä käsikirjoituksen ohjeiden avulla.

H_1: osa löytyneistä ohjelmistovirheistä ei ole löydettävissä pelkästään käsikirjoituksessa määriteltyjen ohjeiden avulla.

Toinen tutkimuskysymys on muotoa: ”*Vaikuttaako testitapausten testausohjeistuksen rakenne ulkopuolisten virheiden esiintymiseen?”* Toiseen kysymykseen saadaan vastaus vertaamalla kevyiden ja raskaiden testitapausten avulla löytyneiden virheiden jakautumista testitapausten sisä- ja ulkopuoliselle alueelle. Hypoteesit toiselle tutkimuskysymykselle ovat:

H_0: testitapausten testausohjeiden rakenne ei vaikuta testitapausten määräämän testikattavuuden sisä- ja ulkopuolisten ohjelmistovirheiden löytymiseen testattavista ohjelmista.

H_1: testitapausten testausohjeiden rakenne vaikuttaa testitapausten määräämän testikattavuuden sisä- ja ulkopuolisten ohjelmistovirheiden löytymiseen testattavista ohjelmista.

5.2. Koeasetelma

Käytetyllä koeasetelmalla pyrittiin ottamaan huomioon mahdollisimman moni kokeeseen vaikuttava muuttuja; kuitenkin jo suunnitteluvaiheessa oli ilmeistä, että kaikkia tämän kaltaisen kokeen tuomia rajoituksia ei kyetä täysin ratkaisemaan sovellettavan koeasetelman puitteissa. Pohdinta vaihtoehtoisista tavoista järjestää tämän kaltainen koe on pyritty esittämään mahdollisimman yksityiskohtaisesti, jotta mahdollisen tulevan tutkimuksen edesauttamiseksi (tästä myös luvussa 7. Tuleva tutkimus).

Kokeeseen valittiin kaksi ohjelmaa, joiden varmistuttiin sisältävän löydettävissä olevia ohjelmistovirheitä, antamalla ohjelmat testattavaksi kahdelle ammattilaistestaajille. Lisäksi, kokeen järjestelyiden kelvollisuus ja soveltuvuus opiskelijoiden osaamistasolle sekä testaukseen suunnitellun ajankäytön sopivuus varmistettiin testaamalla koeasetelmaa kolmella opiskelijalla ennen varsinaisen kokeen järjestämistä. Itse kokeeseen osallistui yhteensä 59 opiskelijaa, joista kaikki olivat tekniikan aloilta – suurimmaksi osaksi tietotekniikasta. Kokeen suorittaminen opiskelijoille kuului osaksi pakollista suoritetta Oulun yliopiston tietotekniikan ohjelmistotekniikka nimisestä kurssista.

Kokeessa käytetyt testitapaukset laadittiin mahdollisimman yhdenmukaisiksi standardin *IEEE 29119: Software and systems engineering-Software testing* kanssa, jotta kokeen käsitteet olisivat selkeät sekä testaus olisi toistettavissa [20]. Tutkimuksessa käytetty termistö puolestaan pohjautuu standardiin *ISO/IEC/IEEE 24765 Systems and software engineering —Vocabulary* vuodelta 2017 [12].

5.2.1. Testattavan ohjelman valinta

Koetta varten valittavalla ohjelmalla voi olettaa olevan huomattava vaikutus kokeen tuloksiin, sillä selvästikin ohjelmistovirheiden olemassaolo testattavasta ohjelmasta on täysin riippuvainen testattavan ohjelman luonteesta ja tilasta. Ohjelman valintaan asetettiin seuraavat alustavat kriteerit:

- opiskelijan on pystyttävä käyttämään ohjelmaa ilman perusteellista perehtymistä siihen,
- ohjelmiston tarkoitus on oltava selkeä,
- ohjelmiston vaatimusmäärittelyt ovat saatavilla testitapausten luomista varten,
- selkeä ja aikarajoitteisiin sopiva kokonaisuus voidaan rajata ohjelmistosta,
- ohjelmistossa on oltava riittävästi ohjelmistovirheitä ja

- ohjelmistosta on saatavilla versio, johon ei tehdä muutoksia kehittäjien toimesta ohjelmiston valinnan jälkeen.

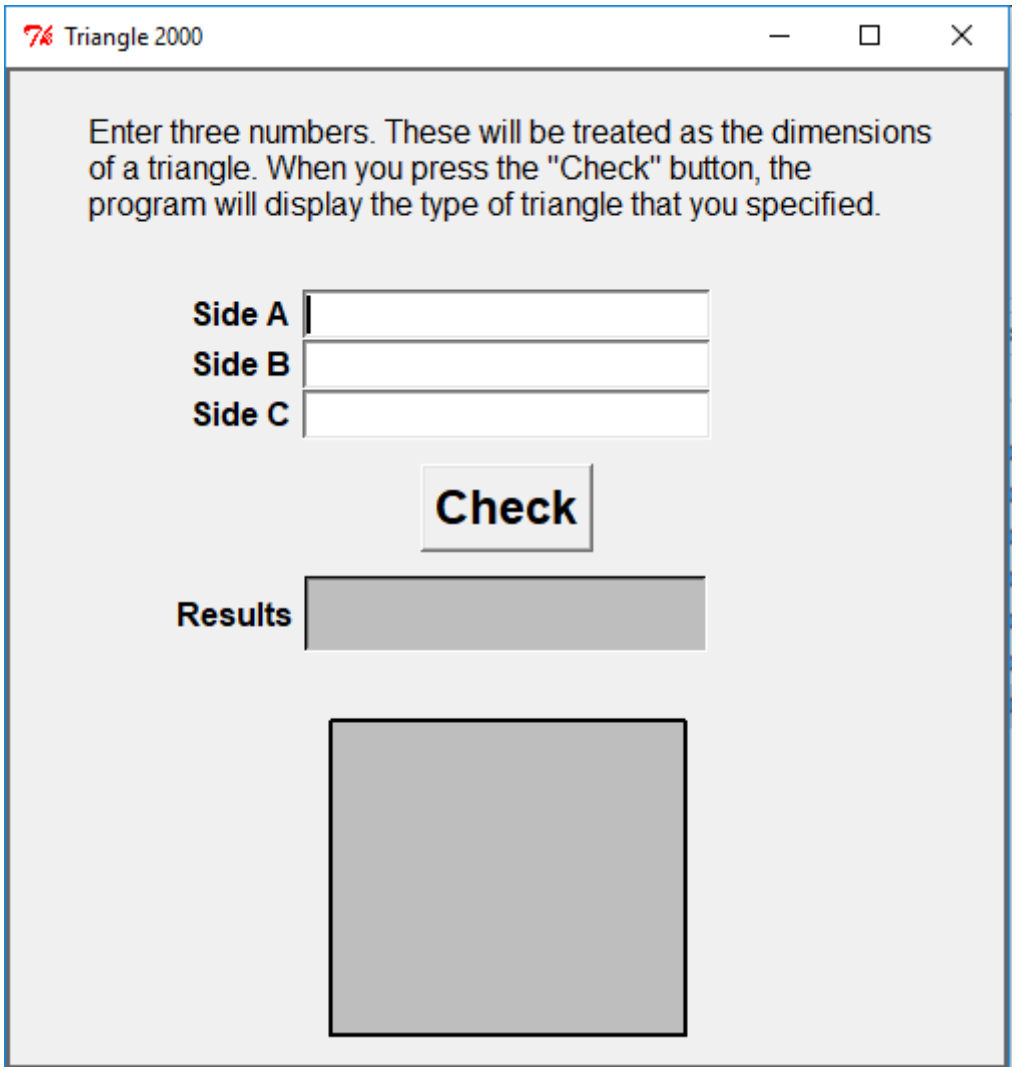
Vaatimusmäärittelyiden saatavuuden kriteeristä jouduttiin luopumaan, sillä tämän toteutuminen osoittautui hyvin haastavaksi. Muutoin valintakriteerit täyttyivät.

Ohjelman etsintä kohdennettiin ensisijaisesti avoimen lähdekoodin ohjelmiin, niiden lisenssien ollessa löyhempiä sekä niiden dokumentaation ollessa helpommin saatavissa. Seulontakriteerit ohjelman selkeästä käyttötarkoituksesta sekä ohjelman soveltuvuudesta opiskelijoiden testattavaksi ilman syvällistä perehtymistä ohjelmaan käytännössä edellyttää ohjelmalta, että se olisi suhteellisen helposti lähestyttävissä keskiverrolle tietokoneen käyttäjälle, ts., käyttäjälle, jolla ei ole teknistä taustaa. Jotta ohjelma olisi lähestyttävä keskiverrolle tietokoneen käyttäjälle; ensiksi ohjelman käyttötarkoitus rajattiin yleishyödyllisiin ohjelmiin ja toiseksi ohjelmasta haettiin selkeää graafista käyttöliittymää. Ohjelmat, joiden käyttötarkoitukset kelpuutettiin tarkempaan tarkasteluun, niiden käyttötarkoituksia olivat mm. kalenteri, työnhallinta, muistikirja, tekstinkäsittely, selain sekä budjetointi. Kaikkiaan näitä ohjelmia pääsi ensimmäisen seulan lävitse kahdeksan kappaletta. Seuraava haaste oli ohjelmiston vaatimusten saatavuus. Tarkasteltujen ohjelmien saataville olevien vaatimusten puutteellisuus osoittautui kuitenkin ongelmalliseksi. Hieman yleistäen, vaatimukset olivat usein karkeita, tulkinnanvaraisia ja epäformaaleja: toiminnallisuuksien vaatimusten esitysmuodot vaihtelivat kehittäjien käymästä dialogista käyttäjien esittämiin toivomuksiin. Useimmille näistä ohjelmista oli saatavilla versioita, joille oli jo entuudestaan suoritettu testausta ja joiden tulokset olivat myös avoimesti saatavilla. Tämän ansiosta ohjelmista oli löydettävissä versioita, joista oli löytynyt ohjelmistovirheitä, ja joiden voitiin todeta olevan korjattuja tai korjaamattomia. Kriteeri ohjelmistovirheiden olemassaolosta saatiin täytetyksi valitsemalla ohjelmasta versio, jossa voitiin todeta olevan ohjelmistovirheitä. Kokeessa käytettäviksi ohjelmiksi valittiin kaksi ohjelmaa: LibreOffice sekä Triangle 2000 [61]. LibreOffice on avoimen lähdekoodin tekstinkäsittelyohjelma ja siitä valittiin beta-testeissä oleva versio, jotta siinä olisi virheitä löydettävissä. Triangle 2000 on puolestaan testauskoulutukseen käytetty ohjelma, minkä vuoksi siinä on virheitä tarkoituksenmukaisesti. Triangle valittiin LibreOfficen rinnalle siltä varalta, että LibreOffice osoittautuisi liian hankalaksi testattavaksi kokemattomille testaajille

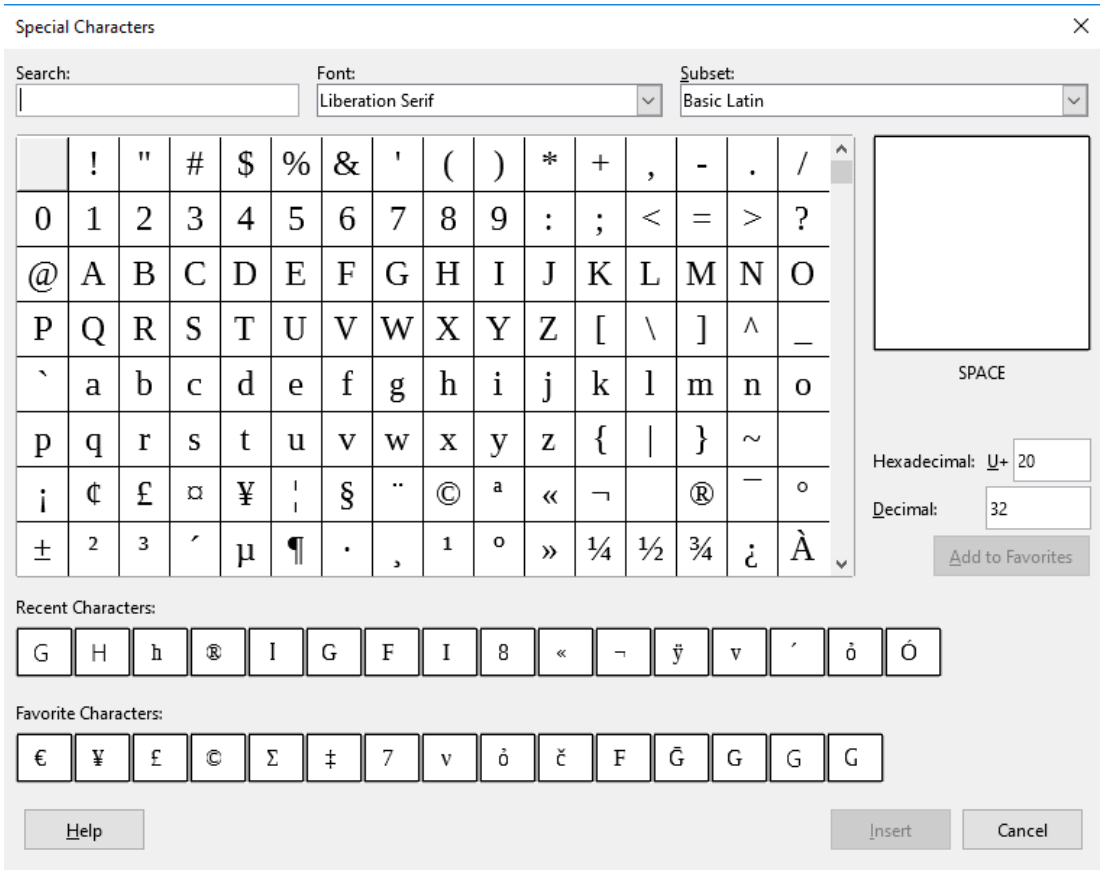
Testattavien ohjelmien valitsemisen jälkeen ohjelmien kokonaisuuksista rajattiin testattavat toiminnallisuudet, jotka sopisivat testauksen aikarajoitukseen. Trianglen tapauksessa koko ohjelma voitiin sisällyttää kokeeseen (kuva 4) ja LibreOfficen tapauksessa pieni selkeästi rajattavissa oleva ominaisuus, erikoismerkkien lisäys (kuva 5), otettiin osaksi koetta. Rajauksen jälkeen testattaville toiminnallisuuksille luotiin rajattujen alueiden toiminnallisuudet kattavat testitapaukset. Tarkastelluilla avoimenlähdekoodin ohjelmilla ei ollut saatavilla formaaleja vaatimusmäärittelyjä, mikä oli myös tilanne valitulle ohjelmalle, LibreOfficelle. Tämän johdosta testitapausten luonnin perustana käytettiin ensisijaisesti ohjelmien käyttöliittymää sekä saatavilla olevia muita materiaaleja, kuten toiminnallisuuksien kuvauksia (sekä Trianglesta että LibreOfficesta oli saatavilla karkeat kuvaukset testattavista kokonaisuuksista). Seuraava aliluku; 5.2.2. Testitapausten suunnittelu, kuvaa testitapausten suunnittelun yksityiskohtaisesti. Ennen valittujen ohjelmien ja niille laadittujen testitapausten lukkoon lyömistä testattavat toiminnallisuudet valituista ohjelmista varmistettiin vielä sisältävän ohjelmistovirheitä antamalla ohjelmat

testattavaksi kahdelle ammattilaistestaajalle (testaajat määrittivät itse sopivakseen katsomansa testauksen yksityiskohdat ja laajuuden), jotka dokumentoivat ohjelmista löytämänsä ohjelmistovirheet. Tämä oli myös ensimmäinen toimenpide sopivan testauksen laajuuden selvittämiseksi koeasetelman aikarajoitteisiin.

Seuraavaksi, opiskelijoille tarkoitetulle koeasetelmalle suoritettiin kaksi erillistä koestusta itse kokeen kulusta kolmella opiskelijalla, joiden tausta oli samankaltainen kokeeseen osallistujien kanssa. Ensimmäisen koestuksen jälkeen testitapauksiin tehtiin parannuksia, jonka jälkeen se toistettiin vielä toisen kerran, jonka jälkeen testitapauksille taas tehtiin hiomista (kirjoitusvirheiden korjaamista, testitapausten järjestämistä ja muuta selkeyttämistä). Näillä toimenpiteillä voitiin tarkastaa koeasetelman toimivuus, testitapausten kelvollisuus sekä opiskelijoille annettavien ohjeiden riittävyys sekä ohjelman soveltuvuus opiskelijoiden testattavaksi.



Kuva 4. Kokeessa käytetty testattu ohjelma Triangle 2000.



Kuva 5. LibreOfficen erikoismerkkien lisäystoiminnallisuus.

5.2.2. Testitapaukset ja niiden suunnittelu

Kokeessa käytettävät testitapaukset luotiin valmiiksi kokeeseen osallistuville. Luodut testitapaukset rajoitettiin LibreOfficen osalta yhteen selkeästi rajattavissa olevaan kokonaisuuteen: erikoismerkkien lisäystoimintoon, jotta testaukseen käytetty aika olisi aikarajoitukseen sopivan laaja. Triangle 2000 on kooltaan pienempi ohjelma, jolloin se voitiin kokonaisuudessaan sisällyttää osaksi koetta. Perustana testitapausten luonnin tukena käytettiin ohjelmien käyttöliittymää sekä saatavilla olevaa muuta materiaalia: kehityksen dokumentaatiota LibreOfficen kohdalla, sekä toiminnallisuuksien kuvauksia Trianglelle. Molempien tapauksessa nämä saatavilla olevat materiaalit olivat aivan liian vähäiset, joten niistä saatu hyöty testitapausten luontiin jäi myös vähäiseksi, jolloin ohjelmien käyttöliittymät toimivat testitapausten luonnin ensisijaisena perustana. Testitapausten luonnin tukena käytettiin testitapausten laatijan osaamista ohjelmistotestauksesta sekä tietoa osasta ohjelmien sisältämistä virheistä (suurin osa virheistä löytyi vasta kokeeseen osallistuneiden toimesta). Vaikka osa virheistä oli tunnettu ennalta, testitapaukset pyrittiin luomaan sellaisiksi, jotka kattaisivat testattavien alueiden kaikki toiminallisuudet kuitenkaan sisältämättä mitään ns. erikoisia testitapauksia; vaikeasti luotavia ja/tai hyvin kompleksisia testitapauksia, elleivät ne olleet oleellisia testattavien perustoiminnallisuuksien kattamiseksi.

Testatuista ohjelmista varmistuttiin, että ohjelmassa olisi virheitä sekä testitapauksien sisäpuolella kuin myös ulkopuolella, ennen kokeen varsinaista

aloittamista ammattilaistestaajien löytämien virheiden, kuin myös koeasetelman testauksen tuloksena. Varsinaisen kokeen tuloksena kuitenkin löytyi vielä paljon ohjelmistovirheitä, joiden olemassaolosta ei tiedetty ennen kokeen järjestämistä.

Testitapaukset rakennettiin kahdella eri suunnitteluperusteella:

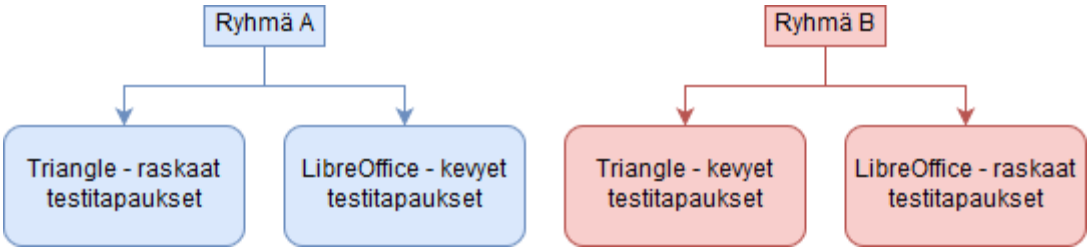
- ns. raskaiksi testitapauksiksi, jotka olivat yksityiskohtaisia kuvauksia testauksen suorituksesta ja tarkoituksesta sekä
- ns. kevyiksi testitapauksiksi, jotka olivat karkeita ja tarkistuslistaa muistuttavia.

Raskaat testitapaukset luotiin testitapauksen mallin mukaiseksi, johon kuuluu testitapauksen kuvaus, tehtävät toimenpiteet (puhekielessä kutsutaan myös *stepeiksi*) ja odotetut tulokset. Kevyet testitapaukset luotiin pitämään sisällään ainoastaan testitapauksen kuvauksen. Kuitenkin neljän kevyen testitapauksen kohdalla katsottiin parhaaksi sisällyttää vaiheet. Näin toimittiin, jotta testauksen aloittaminen varmasti onnistuisi oikeasta ohjelman alueesta LibreOfficen ollessa suhteellisen laaja kokonaisuus, josta oli testattavaksi rajattu vain hyvin spesifi alue. Lisäksi, joidenkin raskaiden testitapausten kohdalla katsottiin paremmaksi ratkaisuksi jättää vaiheet kokonaan pois, sillä näiden testitapausten kuvausten katsottiin olevan itsestään selviä testauksen vaiheista. Kaikki testitapaukset kevyille sekä raskaille testitapauksille luotiin positiivisiksi testitapauksiksi; negatiivisia testitapauksia ei siis käytetty.

Testitapausten lukumäärä kevyille ja raskaille tapauksille ei jakaantunut tasan molempiin luokkiin, sillä raskaiden tapausten suunnitteluperusteen noudattaminen edellytti monen ominaisuuden kohdalla useampaa testitapausta, jotta ominaisuuden perustoiminnallisuus tulisi testatuksi. Tällä jaottelulla testitapausten lukumääräksi tuli A-ryhmälle 35 testitapausta, joista oli 16 raskasta ja 19 kevyttä testitapausta, kun taas B-ryhmällä testitapauksia oli 44, joista 37 raskasta ja 7 kevyttä. Lisäksi raskaat testitapaukset olivat runsassanaisempia: raskaiden testitapausten sanamäärä oli 1898 ja kevyiden 806. Kevyiden ja raskaiden koosta ja testitapausten lukumääräisestä erosta huolimatta, testatut ominaisuudet olivat samat, tai toisin sanoen, jokaiselle testattavalle ominaisuudelle oli kahdet eri testitapaukset: kevyet ja raskaat. Kokeessa käytetyt testitapausten joukot löytyvät liitteistä 1 ja 2: Testitapaukset – ryhmä A ja Testitapaukset – ryhmä B.

Testitapausten testausjärjestyksessä Triangle 2000 asetettiin ennen LibreOfficea, jonka vuoksi toinen ryhmä testasi kevyet testitapaukset ensiksi ja toinen raskaat testitapaukset ensiksi. Osallistujille ei annettu erikseen ohjeita noudattaa juuri tätä annettua järjestystä. Siitä huolimatta, kaikki, yhtä lukuun ottamatta, noudattivat tätä järjestystä ohjelmien testaukseen. Ensimmäisen testitapausten läpikäynnin jälkeen suurin osa kuitenkin vielä palasi aikaisemmin testaamiinsa osioihin. Järjestyksellä, jolla ohjelmat testattiin voi olla vaikutus tuloksiin, kuten myös kevyiden ja raskaiden testausjärjestykselläkin. Triangle 2000 on suoraviivaisempi sekä selkeämpi testattava ohjelma, jolloin tämän testaamisella ensimmäisenä, ohjelmistotestauksesta voi tulla selkeämpi kuva: onhan se juurikin testauskoulutukseen käytetty ohjelma. Tähän ratkaisuun päädyttiin siitäkin huolimatta, että kevyiden ja raskaiden testitapausten testausjärjestyksellä saattaa olla vaikutus kokeen tuloksiin, esimerkiksi: kuinka nopeasti kokeeseen osallistuja sisäistää testauksen päämäärät. Ohjelman katsottiin olevan merkittävämpi tekijä ottaa huomioon testausjärjestystä suunniteltaessa.

Testitapausten jakaminen kahteen osaan siten, että kaikki osallistujat olisivat testanneet sekä kevyillä, että raskailla testitapauksilla, olisi ollut valittujen ohjelmien kanssa haasteellista. Triangle oli kokonaisuutena sen verran pieni, että sen jakaminen kahteen osaan ei olisi onnistunut järkevällä tavalla. Kokeessa käytetyssä jaottelussa päädyttiin siis käyttämään jaottelua, jossa kaikki raskaat testitapaukset yhdellä ryhmällä liittyivät toiseen ohjelmaan ja kevyet toiseen. Tämä on havainnollistettu kuvassa 6.



Kuva 6. Kevyiden ja raskaiden testitapausten jako testaajaryhmille.

5.2.3. Testauksen kulku

Varsinaisen kokeen testaus suoritettiin testitapausten avulla, jotka luotiin valmiiksi osallistujille suunnitteluvaiheessa kokeen laatijan toimesta. Kaikki testausaktiviteetti nauhoitettiin tietokoneen näytöltä koko testaussuorituksen ajalta, jotta saataisiin laaja-alaisempaa informaatiota testaussuorituksista sekä löytyneistä ohjelmistovirheistä. Nauhoituksen avulla voitiin paremmalla varmuudella selvittää, kuinka dokumentoidut ohjelmistovirheet ovat ilmenneet testaajalla; näin ohjelmistovirheen toistaminen ei jäänyt pelkästään testaustulosten tulkinnan varaan. Nauhoitus oli hyödyllinen työkalu virheiden dokumentointien tulkinnassa, mikäli dokumentoinnit eivät olleet yksin riittävän selkeitä. Dokumentaation laatu oli osa-alue, jota ei kokeella ollut tarkoitus tarkastella, ja näin sen vaikutus kyettiin lukemaan pois. Nauhoittamisella saattaa myös olla psykologisia vaikutuksia testaajien suoritukseen: kokeen suorituksen seuranta saattaa esimerkiksi lisätä motivaatiota suorittaa testaus hyvin rangaistuksen pelossa (huono arvosana).

Testaajille annettiin kirjallinen sekä sanallinen ohjeistus testauksesta sekä ohjelmistovirheiden dokumentaatiosta, joka heidän oli tehtävä testauksen lomassa. Dokumentaation ohjeistukseen kuului ohjelmistovirheiden ajankohtien kirjaaminen, ohjelmistovirheiden liittäminen testitapauksiin sekä ohjelmistovirheiden kuvaaminen. Testaajat ohjeistettiin kirjaaman ylös ohjelmistovirheiden löytymisen ajankohdat, jotta ohjelmistovirheiden ilmentymisen ajankohdat olisivat helpommin löydettävissä nauhoituksesta, sekä merkitsemään löytämiinsä ohjelmistovirheisiin minkä testitapausten avulla he ne löysivät.

5.2.3.1. Välineet

Koe toteutettiin tietokoneella, jossa oli Windows 10 -käyttöjärjestelmällä, ja johon oli asennettu Microsoft Word tekstinkäsittelyohjelma dokumentointia varten sekä

Microsoft Excel testitapausten lukua varten. Lisäksi dokumentaatiota varten testaajilla oli käytettävinään ohjelma kuvien ottamiseksi virheistä (Greenshot) [62]. Itse testattavat ohjelmistot olivat valmiiksi asennettuna testausympäristössä. Koe suoritettiin käyttäen yhtä tietokoneen näyttöä. Testauksen nauhoittamista varten testausympäristössä oli käytössä Open Broadcaster Software [63].

5.2.3.2. Kysely testaajille

Testaussuorituksensa jälkeen testaajat täyttivät kyselylomakkeen, jossa selvitettiin testaajien taustaa ja kokemusta. Testaajien taustoista selvitettiin:

- tutkinto-ohjelma,
- kertyneet opintopisteet,
- testauskokemuksen määrän arvio (automaatiotestaus, black-box- ja white-box-testaus eriteltynä),
- ohjelmointikokemus (harrastuneisuus ja opiskelu eriteltynä),
- työkokemus (ohjelmointi ja testaus eriteltynä) sekä
- LibreOffice-ohjelman käyttökokemus.

Osallistujat ilmoittivat opiskeluista saadun ohjelmointikokemuksen suoritetuissa opintopisteissä, jotka sitten muutettiin samaan yksikköön muidenkin arvioitujen työmäärien kanssa työkuukausiksi, jotka laskettiin perusteella 156 tuntia per kuukausi. Ohjelmointi ja testaus kokemuksen määrää on varmasti vaikea arvioida tarkasti, mutta siitä huolimatta niistä voi saada osviittaa kokemuksen määrästä.

5.3. Aineiston rakenne

Kaikki aineisto kerättiin siten, että kokeen järjestäjällä ei ollut mahdollisuutta yhdistää opiskelijoiden suorituksia opiskelijoihin itseensä. Kaikki suoritteet olivat käsiteltävänä satunnaisesti annettujen tunnisteiden takana. Koska kokeen suorite oli osallistujille arvioitava suorite ja sillä oli mahdollinen vaikutus osallistujien kurssin läpipääsyyn ja arvosanaan, opetushenkilöstö tarvitsi tiedon opiskelijoiden suorituksista. Arviointi suoritettiin pelkkien tunnisteiden varassa, jonka tulokset sitten välitettiin kurssin opetushenkilöstölle, joilla oli tieto tunnisteiden takana olevista henkilöistä.

Kokeeseen osallistuneet tuottivat testauksestaan seuraavat materiaalit:

- ohjelmistovirheiden dokumentaation,
- testitapauksille tehdyt kirjaukset,
- testausaktiviteetin nauhoituksen sekä
- kyselylomakkeen vastaukset.

Testauksen dokumentaatiosta kerättiin kaikki virhemerkinnät, jonka jälkeen virheille määrättiin korkean tason virheluokitus. Virheistä muodostettiin aineisto, joka koostui kaikista yksilöllisistä todellisista virheistä sekä yksilöllisistä kuvitelluista virheistä ohjelmassa. Tällä tavalla luokiteltuna kaikkien 588 virhemerkinnän joukossa oli yhteensä 99 uniikkia virhemerkintää (on huomioitava, että tässä lukumäärässä on jonkin verran tulkinnan varaa, riippuen siitä mihin vedetään kunkin virhemerkinnän raja). Toisinaan samasta virheestä oli tehty useampi kirjaus, jolloin useampia virheiden kappaleita saman henkilön kirjaamina käsiteltiin virheiden kaksoiskappaleina (duplikaatteina), eikä siis erillisinä virheinä. Kun 588 virheestä otettiin pois duplikaatit, oli virheiden lukumäärä 523. Virhemerkintöjen viitatessa samaan todelliseen virheeseen tai kuviteltuun virheeseen, ne kirjattiin viittaamaan yhteen ja samaan virheeseen, vaikkakin merkinnät saattoivat vaihdella. Ohjelmien tiedettyjen virheiden perusteellinen tuntemus edesauttoi tässä prosessissa. Väistämättä ohjelmien testatut osat tulivat tunnetuiksi selvitettäessä virhemerkintöjen kelvollisuutta, eli olivatko virhemerkintöjen virheet todellisia virheitä. Kaikki testaajien kirjaamat virheet toistettiin, jotta ne voitiin todeta olevan todellisia virheitä ohjelmissa. Tämä prosessi toteutettiin usealla iteraatiolla. Ensimmäisillä iteraatioilla virheille tehtiin värikoodaus, jossa ne luokiteltiin ryhmiin sen perusteella, mihin ohjelmien ominaisuuksiin ne liittyivät. Kun virheille oli määritelty karkeat luokat, virhemerkintöjä oli huomattavasti helpompi yhdistää saman virheen alle – uniikiksi virheeksi. Kokeeseen osallistujien virheiden dokumentoinnissa yhden virheen kirjaus saattoi sisältää useamman virheen siinä samassa. Näissä tapauksissa virheet eriteltiin kokeen tuloksissa. Mikäli virhemerkintä ei täyttänyt virheen kriteerejä, mutta se selkeästi sisälsi oleellista palautetta ohjelman toiminnasta, merkinnät luokiteltiin luokkaan ”ei ole virhe”. Esimerkkinä tästä olisi toiminto, joka testaajan mielestä olisi hyvä olla ohjelmaan sisällytettynä tai ohjelman jonkin toteutuksen ollessa epäoptimaalinen testaajan näkökulmasta. Pääsääntöisesti tällaiset virhemerkinnät luokiteltiin luokkaan ”ei ole virhe”. Tällainen informaatio voi olla arvokasta palautetta ohjelman toiminnasta ja kaikista selkeimmät hyvät huomiot luokiteltiin virheiksi, sillä perusteella, että kyseinen toiminto olisi hyvä ohjelmasta muuttaa testaajan palautteen perusteella.

Alkuperäiset tavoitteet kokeen tuottamasta informaatiosta kokivat sen suorituksen aikana pieniä muutoksia. Eritellyille uniikeille virheille oli tarkoitus määrittää ulkopuolisuus suhteessa testitapauksiin (joko ulkopuolinen tai sisäpuolinen), kelpoisuus (ts. onko virhemerkintä todella virhe), virheiden vakavuus ja virheiden tyyppi. Löytyneiden virheiden tyyppien määrittäminen osoittautui haasteelliseksi (virheen luokitteleminen toiminnallisuusvirheeksi ei ole täysin yksiselitteistä, sillä se voi olla myös samanaikaisesti esim. käytettävyysvirhe) ja kaiken lisäksi ohjelmista löytyneet virheet olivat lähes täysin toiminnallisuus- ja käytettävyysvirheitä. Tämän vuoksi tyyppiluokituksesta päätettiin luopua kokonaan. Samankaltainen ongelma oli suunnitellun IEEE-vakavuusluokituksen (merkityksetön, vähäinen, vakava ja kriittinen) kanssa [64]. Sen käyttäminen oli haasteellista käytettyjen ohjelmien luonteen sekä niiden keskinäisen eroavaisuutensa tähden. Trianglen testatut ominaisuudet olivat keskeisimpiä kuin LibreOfficella, jonka testatuissa ominaisuuksissa ei ollut läheskään niin suurta keskeisten ominaisuuksien osuutta. IEEE-vakavuusluokituksen sijasta käytettiin suhteellista asteikkoa, jossa virheet laitettiin suhteelliselle asteikolle, jossa kaikki löytyneet virheet laitettiin neljän vakavuusluokan vakavuusjärjestykseen. Lisäksi, kelpoisuus yhdistettiin yhdeksi

asteikoksi vakavuuden kanssa, jossa vakavuuden arvo 0 tarkoittaa virhemerkintää, joka ei viittaa todelliseen virheeseen. Määritettäessä ulkopuolisuuden tasoa virheille, havaittiin, että virheille voisi määrittää ulkopuolisuuden taso tarkemmin. Sisäpuoliset virheet jaettiin eksplisiittisiin virheisiin ja implisiittisiin virheisiin ja ulkopuoliset virheet jaettiin liittyviin virheisiin ja täysin ulkopuolisiin virheisiin. Tämän kaltaisessa luokituksessa on myös se etu, että virheet voidaan luokitella ordinaalisesti suhteessa testitapauksiin (täysin ulkopuolinen virhe on kauempana testitapauksen eksplisiittistä kuvausta kuin liittyvä virhe jne.).

Näin käsiteltynä uniikeille virhemerkinnöille määrättiin seuraavat ominaisuudet:

- kelpoisuus, joka määrittää ohjelmistovirheen kriteerien täyttymisen;
- vakavuus (asteikolla 1-4);
- ulkopuolisuuden taso, tai toisin sanoen etäisyys testitapauksista (0-3).

Näistä kelpoisuus ja vakavuus kuvattiin samalla asteikolla, jossa 0 on ”virhemerkintä ei ole virhe” ja arvot 1-4 ovat ”virhemerkintä on virhe” ja asteikon arvo kuvaa virheen suhteellista vakavuutta kaikkiin löytyneisiin virheisiin. Ulkopuolisuudella tarkoitetaan tässä etäisyyttä testitapauksen kirjalliseen asuun.

Virhemerkinnöille määrättiin neljä ordinaalista ulkopuolisuuden tasoa suhteessa testitapauksiin:

- eksplisiittiset (0),
- implisiittiset (1),
- liittyvät (2) ja
- täysin ulkopuoliset (3).

Tässä neljän tason jaottelussa:

- eksplisiittisille virhemerkinnöille voidaan vetää *suora yhteys* testitapauksen eksplisiittiseen sisältöön,
- implisiittisille virhemerkinnöille voidaan vetää *epäsuora yhteys* testitapausten sisältöön (ts. testitapaus ei välttämättä ota suoraa kantaan sisältöön, mutta käytännössä se on kuitenkin sisältyneenä testitapaukseen),
- liittyville virhemerkinnöille *ei voida vetää yhteyttä* testitapauksiin, mutta virhemerkinnän sisältö kuitenkin liittyy testitapaukseen, kun taas
- täysin ulkopuolisille virheille ei voida vetää *ollenkaan yhteyttä* testitapauksiin.

Ulkopuolisuuden tasot ovat esitettynä tarkemmin tuloksissa.

Virheiden vakavuuden luokittelu tehtiin antamalla virheille vakavuuden suhteellinen taso 1-4, jolloin kaikki löytyneet virheet jaettiin tasan näille eri tasoille. Näin jokaisella tasolla oli 25% kaikista kelvollisista virheistä (15 virhettä per taso). Tämä ei välttämättä täysin tuo esille vähäpätöisimmän virheen ja vakavimman virheen mahdollista suurta eroa, mutta on kuitenkin suuntaa antava. Virhemerkinnöille, jotka eivät olleet todellisia virheitä, vakavuudeksi asetettiin 0. Vakavuus luokittelussa

käytettiin seuraavia kriteereitä: estää ohjelman käytön; toiminnallisuus on epälooginen ohjelman tarkoituksen kontekstissa; haittaa tai estää toiminnallisuuden tarkoituksen toteutumista; virhe on keskeisessä toiminnallisuudessa ja haittaa ohjelman käyttöä. Prosessissa arvioitiin virhemerkintöjen alla olevien virheiden ominaisuuksia – ei siis virhemerkintöjen (arviointi tehtiin uniikeille virheille). Virhemerkintä ei välttämättä tuo esille virhemerkinnän vakavuutta samalla tavoin kuin virhe tarkasteltuna virhemerkinnän ulkopuolella, jos virhemerkintä on esimerkiksi epäselvä tai ylenpalttisen dramaattinen.

5.4. Tarkkuus ja palautus

Tarkkuus ja palautus (engl. precision and recall) ovat kaksi tiedonhaun yhteydessä käytettyä metriikkaa, jotka kuvaavat tiedonhaun tehokkuutta [65, s. 154]. F-arvo puolestaan on metriikka, joka yhdistää nämä kaksi metriikkaa [66]. F-arvoa on aiemminkin käytetty ohjelmistoalan tutkimuksen yhteydessä, mm. Mäntylä ja Itkonen 2013 [67]. F_1-arvon (englanniksi F_1-measure) laskukaava on:

$$F = 2 * \frac{tarkkuus * palautus}{tarkkuus + palautus}, \quad (1)$$

jossa tarkkuuden ja palautuksen kaavat ovat:

$$tarkkuus = \frac{tp}{tp + fp} \quad (2)$$

$$palautus = \frac{tp}{tp + fn} \quad (3)$$

ja jossa $tp = true\ positive$, $fp = false\ positive$ ja $fn = false\ negative$. Tämän kokeen kontekstissa true positive on virhemerkintä, joka on todellinen virhe, false positive on virhemerkintä, joka ei ole virhe, ja false negative on virhe, jota ei löytynyt testaajan toimesta. Tarkkuus kuvaa kuinka moni kirjattu virhemerkintä on todella virhe ja palautus taas kuvaa kuinka suuri osuus ohjelmissa olevista virheistä löytyi. F-arvon laskentakaavassa on oletettu, että kaikki ohjelmissa olevat virheet olisi tunnettu (tai tarkemmin sanottuna ainakin riittävän lähellä tätä). Ennen kokeen järjestämistä ohjelmat annettiin testattavaksi ammattilaistestaajille, jotta saataisiin selvitettyä ohjelmissa olevan virheitä. Kaikki nämä kyseiset virheet löydettiin myös kokeeseen osallistujien toimesta ja vieläpä niiden lisäksi monin verroin enemmän virheitä. Tämän perusteella on katsottu perustelluksi tehdä oletus, että virheet, jotka kokeeseen osallistujat löysivät, on riittävän lähellä niiden virheiden lukumäärä, jotka olisi tällä lähestymistavalla löydettävissä ohjelmista, ja siten F-arvon laskentakaavan soveltuvan kokeen aineistolle. Toki on mahdollista, että vielä löytyisi lisää virheitä, esim., kompleksisempia virheitä, muutoin jotain erityisosaamista vaativaa tai aivan toista lähestymistapaa testaukseen (esim. white-box testaus). Oletettavasti näin myös onkin, mutta sen ei pitäisi vaikuttaa F-arvon soveltuvuuteen, sillä näiden virheiden olemassa-

ololla olisi yhtä suuri suhteellinen vaikutus laskettuihin F-arvoihin. Ne muuttaisivat ainoastaan F-arvojen absoluuttisia arvoja, mutta ei niiden suhteellisia arvoja toisiinsa.

Käytetyllä ohjelmalla oli merkittävä vaikutus F_1-arvoon, jossa ohjelman koko on keskeinen tekijä: LibreOfficella oli huomattavista suurempi määrä virheitä (43 virhettä) kuin Trianglella (17 virhettä). F_1-arvoon vaikuttaa löytyneiden virheiden suhteellinen osuus kaikista virheistä, joita ohjelmissa oli (tarkkuuden kaavassa: $tp + fp$). F_1-arvon voisi laskea joko käyttämällä yhteistä virheiden kokonaismäärää ($43 + 17 = 60$) tai erillisinä lukuina: 17 ja 43. Viiden virheen löytäminen tuottaa huomattavasti suuremman tarkkuuden Trianglesta kuin LibreOfficesta, jolla on suuri vaikutus F_1-arvoon. Tämän johdosta F_1-arvot Libren ja Trianglen välillä eivät ole täysin vertailukelpoisia käytettäessä molempien omaa virheiden kokonaislukumäärää. Koska virheiden löytäminen on ajallisesti kriittistä, LibreOfficesta ei edes olisi mahdollista löytää samaa suhteellista osuutta kaikista ohjelman virheistä, mistä puolestaan seuraa, että F_1-arvot ohjelmien välillä eivät ole täysin vertailukelpoisia. Mikäli pitäydyttäisiin F_1-arvossa, joka lasketaan kahdesta erillisestä virheiden kokonaislukumäärästä, olisi tarpeellista käyttää suhteellisesti yhtäläinen aika samaan määrään virheitä (testaukseen, kuin myös virheiden dokumentoimiseen kuluu oma aikansa). Sen huomioimiseksi, virheiden F_1-arvot on laskettu sekä yhteisestä virheiden kokonaislukumäärästä [60], että erillisinä (17 ja 43), jolloin testaukseen käytetty aika on paremmin otettu huomioon F_1-arvoja laskettaessa.

F_1-arvon lisäksi laskettiin F_β-arvot β:n arvolla 2 (F_1-arvon β on 1). F_1-arvossa sekä tarkkuudella, että palautuksella on sama painoarvo, mutta se voidaan myös määrittää toisin, mikäli se katsotaan tarpeelliseksi F-arvoa sovellettaessa [66]. F_β-arvon laskenta eroaa F-arvosta asettamalla sen laskentakaavaan β-luku, jolla voidaan asettaa eri painoarvo tarkkuudelle ja palautukselle. Tällöin F_β-arvon kaava on:

$$F = (1 + \beta^2) * \frac{tarkkuus * palautus}{\beta^2 * tarkkuus + palautus}. \quad (4)$$

Koska testauksessa väärän virheen kirjaamisen voisi katsoa olevan vähemmän vakavaa kuin virheen löytymättä jääminen. Perinteisen F-arvon ja F_β-arvon välillä ei kuitenkaan β:n arvolla 2 ollut merkittävää eroa käytettäessä Wilcoxonin järjestyssummatestiä. Tämän vuoksi analyysissä kuitenkin pitäydyttiin F_1-arvossa.

5.5. Rajoitukset

Kokeen järjestelyihin liittyen on tunnistettavissa lukuisia potentiaalisia vaikuttavia tekijöitä sekä rajoitteita. Näitä voidaan tunnistaa eritoten testitapauksista, testitapausten laadinnasta, testattavista komponenteista, testattavasta ohjelmistosta sekä kokeeseen osallistujista. Mahdolliset vaikuttavat tekijät pyrittiin jo huomioimaan kokeen järjestelyillä. Samanlaista koetta ei ole aikaisemmin tehty – ainakaan tällaista koetta ei löydetty tämän opinnäytetyön tutkimuskatsauksen yhteydessä. Samankaltainen koejärjestely kylläkin on eri tarkoituksella tehty [11, 68].

Kokeen testitapausten laatijan ollessa yksittäinen henkilö, laatijan taitotaso ja muut yksilölliset ominaisuudet voivat heijastua testitapausten laatuun tai määrään, kuin

myös virhemerkintöjen tulkintaan. Näillä taas voi olla vaikutus niin testitapausten sisä- kuin ulkopuolistenkin ohjelmistovirheiden löytymiseen ja luokitteluun.

Sillä käytössä oli vain kaksi testitapausten joukkoa ja kaksi ohjelmaa, on tarkastelun kohteena kaksi kiintopistettä kustakin poikkeamien esiintymiselle. Jos puhutaan yleisesti testauksesta, on väistämättä samalla puhuttava yleisestä ohjelmasta ja yleisistä testitapauksista. Tarkastelemalla vain kahta ohjelmaa ja kahta testitapausten joukkoa, tuskin voidaan väittää näiden vastaavan yleistä ohjelmaa ja yleisiä testitapauksia. Keskeinen kysymys on: ovatko käytetyt testitapaukset ja ohjelmat riittävän lähellä yleistä. Käytetyillä ohjelmilla ja testitapauksilla on varmasti sellainen vaikutus kokeen tuloksiin, että vaikka kaikki muut tekijät huomioitaisiin, käytetyt ohjelmat ja testitapaukset eivät vastaa yleistä.

5.5.1. Testitapaukset

Testitapaukset suunniteltiin, jotta ohjelman kaikki toiminnallisuudet katettaisiin testitapauksilla. Testitapaukset muotoiltiin, jotta osalla niistä löytyisi ohjelmistovirheitä ja osalla ei (osa virheistä oli tunnettu koetta valmisteltaessa). Sen lisäksi testitapausten ulkopuolelle jätettiin tarkoituksella virheitä. Osa näistä testitapausten ulkopuolisista virheistä jätettiin selkeästi testitapausten ulkopuolelle ja osa taas lähemmäksi testitapausten kattamaa aluetta.

Testitapausten ominaisuuksilla saattaa olla vaikutus. Vaikka pyrkimyksenä oli tukeutua mahdollisimman paljon standardeihin, testitapausten luonnin olosuhteet (mm. saatavilla olevat vaatimusmäärittelyt ja ohjelmien rakenne) vaikuttivat testitapausten lopulliseen rakenteeseen. Luoduilla testitapauksilla pyrittiin kattamaan varattujen ohjelmien perustoiminnallisuudet. Tämä ei välttämättä vastaa täysin testitapauksia, jotka olisi luotu kyseisiä ohjelmia varten. On mahdollista, että testitapaukset, jotka laadittaisin kyseisille ohjelmille, sisältäisivät erikoisempiakin testitapauksia, vaikka perustoiminnallisuuksien testaus ei tätä edellyttäisi.

Testitapaukset olivat kirjoitettu englannin kielellä, kun kaikki testaajat olivat suorittamassa suomen kielistä tutkintoa. Testaajien englannin kielitaidossa on voinut olla eroavaisuuksia. Testitapausten kielivalinnalla voi olla vaikutus kokeen tuloksiin; esimerkiksi, jos testaaja joutuu tukeutumaan sanakirjaan testauksen ohella, testaus prosessi eittämättä hidastuu. Lisäksi, testitapaukset olivat Excelissä ja dokumentaatio oli tehtävä Word-dokumenttiin, näiden ohjelmien hallitsemisella voi olla vaikutus testauksen kulkuun.

Jos taas kokeessa olisi vaihtoehtoisesti jokaisen testaajan itse luoma testitapausten joukko, olisi yleistysten tekeminen hankalampaa, sillä eri testitapausten joukkojen välillä olisi auttamatta laadullista poikkeamaa, mikä tulisi myös ottaa huomioon. Jälkimmäisessä tilanteessa, testaaja auttamatta perehtyisi testattaviin ohjelman ominaisuuksiin jo ennen testausta sekä henkilön itse luomat testitapaukset voisi olettaa olevan hänelle itselleen ymmärrettävämpiä kuin toisen henkilön luomat testitapaukset. Täten molemmilla lähestymistavoilla voi katsoa olevan omat etunsa ja haittansa.

Testitapausten luonnin perusteena käytettiin toiminnallisuuksien määrittelyjä, jotta ne voitaisiin määrittää eksplisiittisesti testattavien toiminnallisuuksien määrittelyiden pohjalta. Tarkkojen vaatimusmäärittelyiden puuttuessa, testitapausten luontiin käytettiin testausta varten saatavilla olevaa materiaalia koostuen toiminnallisuuksien kuvauksista, kehityksen dokumentaatiota sekä julkaisumuistiosta. Tämän johdosta

paremman testikattavuuden saavuttaminen olisi testitapausten laatijan tulkinnan ja taidon varassa. Testitapausten luontiin käytettyä aikaa ei myöskään erikseen rajoitettu. Sen sijaan, testitapausten luonnin rajoitteena käytettiin saatavilla olevan dokumentaation kattavuutta testattavien ominaisuuksien suhteen.

Kevyiden testitapausten epämääräisyys voi käytetyssä ulkopuolisuuden tasojen luokittelussa johtaa suurempaan testitapausten kattamaan alueeseen. Implisiittinen alue on suurempi, mikäli testitapaus ohjeistaa testaamaan jonkin ohjelman osion kokonaisuudessaan, kuin, jos testitapaus ohjeistaisi testaamaan ohjelman osiosta osasia, on testitapausten eksplisiittinen alue pienempi ja siten ulkopuolisuuden tasojen jakauma erisuuri. Täten ulkopuolisuuden tason käsite on täysin riippuvainen käytetyistä testitapauksista.

Mikäli käytetään useampaa ohjelmaa, olisi hyvä saada testattavat ohjelmat mahdollisimman saman laajuiseksi testattavuudeltaan. Ohjelmien poikkeavuudet voivat näkyä tuloksissa. Järjestettäessä koeasetelma yhdelle ohjelmalle, vaihtoehto järjestää kokeen ohjelmien testaukset erillisinä kokonaisuuksina tämän sijaan. Useamman ohjelman käyttämisessä koeasetelmassa voi olla etunsa; jos puhutaan yleisesti testauksesta, eikä vain jonkin tietyn ohjelman testauksesta, on myös oltava ns. yleinen ohjelma. Voisi olettaa, että kaksi ohjelmaa on todennäköisemmin lähempänä tätä yleistä ohjelmaa, kuin yksi ohjelma. LibreOfficen käyttöliittymä oli huomattavasti monimutkaisempi ja sen käyttäminen oli voimakkaammin käyttöliittymään sidonnainen. Triangle puolestaan paljon matemaattisempi.

5.5.2. Virheet

Samoin kuin testitapausten luonti, kaikki virheiden luokittaminen tehtiin kokeen järjestäjän toimesta, millä voi olla vaikutus virheiden luokitteluun. Optimaalisessa tilanteessa ohjelmiston asianosainen voisi luokitella löytyneet virheet vakavuuden osalta, hänen tuntiessa ohjelman ja sen prioriteetit paremmin. Vaihtoehtoisesti osan luokittelusta olisi voinut jättää testaajan kontolle, mutta testaajat tekivät jo virhemerkintöjä tehdessään virheitä, joten tästä voisi tehdä johtopäätöksen, että he olisivat myös tehneet virheitä tekemässään luokittelussa. Mikäli testaajilla olisi ollut enemmän kokemusta testauksesta; ehkä sitten testaajille olisi voinut antaa enemmän luokitteluvastuuta. Koeasetelmassa testaajilla voisi myös katsoa olevan intressi luokitella omat löydöksensä merkittävämmäksi kuin ne todella ovat, sillä testaajalla ei olisi ollut mitään menetettävää suorituksensa yliarvioinnista.

Osa osallistujista oli yhdistellyt läheisiä virheitä saman virhedokumentaation alle, kun taas osa oli jättänyt yhdistelemättä. Tämä huomioitiin, kun virheet luokiteltiin uniikkien virheiden eikä virhemerkintöjen mukaan. Tällaisten virheiden kohdalla virheitä kohdeltiin yhdenmukaisesti: löytyneet virheet jaettiin joko useampaan osaan tai yhdistettiin yhdeksi virheeksi. Tämä toimenpide aiheutti pieniä muutoksia virheiden kokonaislukumäärään.

Osallistujien kirjaamille virheille oli useita eri esitysmuotoja, joista osa parempia kuin toiset. Virheiden aiheuttajien paikantaminen ei onnistunut kaikilta täydellisesti, mutta sitä ei katsottu ongelmaksi kokeen tuloksissa, sillä jo virheeseen osoittaminen itsessään voisi katsoa olevan ohjelmoijalle hyvä johtolanka, jota hän voisi lähteä seuraamaan virheen korjaamiseksi. Jotkin virhemerkinnät olivat niin epäselviä, että ne eivät olisi riittäneet niissä käsiteltyjen virheiden löytämiseksi ja kyseiset virheet

kävivät vasta nauhoituksesta ilmi. On siis huomioitava, että tällä järjestelyllä voidaan tarkastella: löysikö testaaja virhettä, eikä sitä osaako testaaja dokumentoida virheitä tai tekikö testaaja dokumentaatiossaan virheitä (esim. merkitsi virheen väärän testitapauksen kohdalle). Testitapaukset saattavat vaikuttaa testausdokumentaation laatuun, mutta sitä ei tarkasteltu tällä kokeella. Koeasetelma ei myöskään ota huomioon dokumentaatioon menevää aikaa täysin. Teoriassa, osallistujalla oli mahdollisuus päästä parempiin lopputuloksiin tässä koeasetelmassa, mikäli hän teki heikon dokumentaation, kuin että olisi tehnyt hyvän dokumentaation. Sillä hyvän dokumentaation tekemiseen menee kauemmin aikaa kuin heikon dokumentaation, jolloin heikon dokumentaation tekijä pystyi käyttämään enemmän aikaa testaukseen dokumentaation sijasta.

5.5.3. Koe

Oikeaa osallistujien satunnaistamista ei tehty – jako ryhmiin tehtiin ensisijaisesti osallistujien ilmoittautumisen mukaan eri harjoitusryhmiin, joissa koe suoritettiin. Koe suoritettiin neljässä erässä, neljänä eri päivänä ja neljässä eri tietokoneluokassa. Tällöin esimerkiksi kellonajalla (aamulla osallistujat saattavat olla virkeämpiä), tilalla tai jollain muulla ryhmäjaottelun aiheuttamalla tekijällä saattoi olla vaikutus tuloksiin. Yhden ryhmän kanssa verkkoyhteys aiheutti ongelmia kokeeseen: kokeen aikana oli noin 5 minuutin yhteyskatkos. Koeasetelmaa suunniteltaessa ajateltiin, että kaikki osallistujat käyttäisivät testaukseen määrätyn ajan, mutta näin ei aivan kuitenkaan käynyt ja testaukseen käytetty aika hieman vaihteli testaajien välillä. Koska testaus suorituksista oli tallennettuna nauhoitukset; jokaiselle testaajille oli mahdollista määrittää testaukseen käytetty aika tarkasti katsomalla nauhoituksesta testauksen aloitus- ja lopetusajankohdat. Pitkästyminen on tekijä, joka on saattanut vaikuttaa tuloksiin, mutta toisaalta tieto hyvien suoritusten palkitsemisesta on saattanut vähentää pitkästymisen vaikutusta (20% parhaiten kokeessa menestyneet palkittiin).

Sillä, että opiskelijoiden testaus nauhoitettiin, voi olla vaikutus näiden tekemään testaukseen. Ihmisen kokiessa hänen olevan tarkastelun alla, hän saattaa toimia poikkeavasti verrattuna hänen toimiessa ilman tarkastelua. Lisäksi, kokeeseen osallistujien taustalla voi olla vaikutus kokeen tuloksiin, osallistujien ollessa testauksesta kokemattomia opiskelijoita, joilla taustana oli ainoastaan teoreettinen tausta testaukseen. Mikäli sama koe tehtäisiin esim. ammattilaistestaajilla, tulokset voisivat poiketa. Opiskelijoiden testaus ei siis välttämättä vastaa ammattilaisten testausta, jota loppujen lopuksi harjoitetaan ohjelmistoteollisuudessa.

Yhteenveto koeasetelman rajoituksista on esitetty taulukossa 1. Kyseisessä taulukossa koeasetelman rajoitukset on jaoteltu neljään luokkaan: virheisiin, testitapauksiin, osallistujiin ja aineiston keruuseen liittyviin rajoituksiin.

Taulukko 1. Yhteenveto rajoituksista.

Virheet	Testitapaukset	Osallistujat	Aineiston keruu
Monet luokittelu perusteet eivät ole täysin ehdottomia: rajatapauksia esiintyy.	Kokeessa käytetyt testitapaukset eivät ole otos kaikista testitapauksista, joita saatetaan käyttää.	Opiskelijoiden käyttö oikeiden testaajien sijasta: kokemuksen puute.	Virhemerkinnät toisinaan epäselviä ja testaajat tekivät paljon merkintä virheitä: kaksinkertainen kirjaus testitapauksiin, virhe dokumenttiin sekä nauhoitukseen tukeutuminen toivat näitä esille.
Ulkopuolisen tekemä luokittelu vs. testaajan itse tekemä luokittelu. Molemmilla on hyvät ja huonot puolensa.	Kokeessa käytetyillä ohjelmilla voi olla merkittävä vaikutus kokeen tuloksiin: yhden ohjelman käyttäminen tekisi kokeen järjestämisestä selkeämpää.	Testaustilanne: vaikka osallistujilla oli motivaatio suoriutua testauksesta hyvin: testaustilanne ei vastaa normaalia testausasetelmaa.	Aineiston keruu työläs ja virhealtis. Virheitä oli korjattava usealla aineiston tarkistuksella.
Virheiden luokitteleminen useampaan luokkaan.	Testitapausten testausjärjestys: aloittaako testaaja testaamaan kevyillä vaiko raskailla, voi vaikuttaa testauksen tuloksiin.	Osallistujien satunnaistamisen puute.	Jos puhutaan yleisesti testauksesta, on samalla puhuttava yleisestä ohjelmasta ja yleisistä testitapauksista. Tarkastelemalla vain kahta ohjelmaa ja kahta testitapausten joukkoa, tuskin voidaan väittää näiden vastaavan yleistä ohjelmaa ja yleisiä testitapauksia.
Dokumentaation ja testitapausten suhdetta ei tarkasteltu.		Osallistujien pitkästyminen on mahdollinen vaikuttaja tuloksiin.	
		Opiskelijoiden käyttäminen testaustutkimuksessa.	
		Ajankäyttö testaussuorituksista ei ollut yhtä pitkä kaikilla.	

6. TULOKSET

Johtuen koeasetelman rakenteesta ja sen muuttujista sekä kokeen rajoituksista (5.5. Rajoitukset) katsottiin oleelliseksi laskea ohjelmien ja ryhmien mahdollinen vaikutus kokeen tuloksiin. Näin toimittiin erityisesti siksi, että vastaavanlaista koetta ei löydetty työn taustatutkimuksen yhteydessä, jolloin asetelma oli uusi ja sen toteutus itsessään kokeellinen. Tutkimuskysymyksiin vastaamiseksi ja testikattavuuden ulkopuolisen alueen tarkastelemiseksi määritettiin ulkopuolisuuden tasot:

- eksplisiittinen,
- implisiittinen,
- liittyvä ja
- täysin ulkopuolinen taso.

Kokeen tarkasteltujen muuttujien tilastollista eroavuutta testattiin Wilcoxonin järjestyssummatestillä, joka on parametriton menetelmä ja siten ei edellytä tiettyä jakaumaa testattavalta aineistolta. *Testiin liittyvän p-arvon avulla voidaan arvioida, ovatko jakaumat erilaiset ja r-suureen avulla selviää eron suunta: positiivisilla arvoilla ensimmäinen ryhmä on suurempi ja negatiivisissa tapauksissa taas toinen esitetty ryhmä.*

6.1. Ohjelmien ja ryhmien vaikutus

Kokeeseen osallistujat jaettiin kahteen ryhmään, joista toinen ryhmä testasi Trianglen raskailla testitapauksilla ja LibreOfficen kevyillä testitapauksilla (ryhmä A) ja toinen ryhmä puolestaan testasi Trianglen kevyillä testitapauksilla ja LibreOfficen raskailla testitapauksilla (ryhmä B). Arvioitaessa kevyiden ja raskaiden testitapausten eroa testitapaukset laadittiin muotoonsa, jotta molemmilla ryhmillä olisi samankaltaiset kevyet ja raskaat testitapaukset riippumatta testatusta ohjelmasta.

Ryhmien välillä ei ollut tilastollisesti merkitsevää eroa virheiden lukumäärän ($p = 0{,}366$, $r = 0{,}122$), vakavuuden ($p = 0{,}167$, $r = 0{,}186$), virheellisten virhemerkintöjen lukumäärän ($p = 0{,}251$, $r = -0{,}154$) tai F-arvojen suhteen ($p = 0{,}218$, $r = 0{,}166$) (taulukko 2). Virheiden etäisyydestä testitapauksiin oli kuitenkin merkittävä ero A:n ja B:n välillä: A-ryhmän virheiden etäisyys oli B:tä suurempi ($p < 0{,}001$, $r = 0{,}524$) (taulukko 2). A-ryhmä testasi Trianglen raskailla testitapauksilla ja LibreOfficen kevyillä testitapauksilla. Kaikkiaan Trianglen virheet olivat lähempänä testitapauksia LibreOfficeen verrattuna ($p = 0{,}011$, $r = -0{,}241$) ja LibreOfficen kevyiden testitapausten etäisyys ($p < 0{,}001$ ja $r = 0{,}782$) olikin Trianglen vastaavia suurempi ($p < 0{,}001$, $r = 0{,}573$).

Taulukko 2. Ryhmien (A ja B) välisen Wilcoxonin järjestyssummatestin tulokset tarkastelluille muuttujille.

Muuttuja	Testi-tapaukset	Mediaani (sd)	*p*-arvo	W	r
Virheiden lukumäärä	Ryhmä A $n = 26$	8 (2,86)	0,366	445	0,122
	Ryhmä B $n = 30$	7 (2,50)			
Vakavuus	Ryhmä A $n = 26$	24,5 (8,37)	0,167	474,5	0,186
	Ryhmä B $n = 30$	19,5 (7,81)			
F-arvo	Ryhmä A $n = 26$	0,232 (0,0712)	0,2175	465,5	0,166
	Ryhmä B $n = 30$	0,206 (0,0660)			
Väärät virhe-merkinnät	Ryhmä A $n = 26$	1 (0,86)	0,2514	321,5	-0,154
	Ryhmä B $n = 30$	1 (1,07)			
Ulko-puolisuus	Ryhmä A $n = 26$	1,1464 (-0,3102)	< 0,001	628,5	0,524
	Ryhmä B $n = 30$	0,6333 (0,3749)			

Osa testaajista epäröi testauksen aloittamista heidän vielä lukiessa ohjeita ohjeistuksen antamisen jälkeenkin, jolloin heidän testauksensa aloittaminen hieman viivästyi. Osa taas lopetti testauksen etuajassa: testattuaan testitapaukset kerran lävitse eivät he enää jatkaneet testausta tämän jälkeen. Vaikka mediaani ajankäytölle oli koko testaukseen varattu aika eli 120 minuuttia, moni lopetti testauksen selkeästi ennen ajan päättymistä: neljäsosa osallistujista lopetti testauksen 80-108 minuutin kohdalla testauksen aloittamisesta. Kesken jättäneiden nauhoituksista oli havaittavissa, että testitapausten loppuessa kesken myös testaus loppui siihen paikkaan. Kuitenkin osa testaajista vielä jatkoi testaamista testitapausten loppumisenkin jälkeen mm. tarkastamalla tekemiänsä tuloksia tai testaamalla uudestaan jo testattuja testitapauksia/niissä käsiteltyjä ominaisuuksia. Joukkoon mahtui myös testaajia, jotka eivät ennättäneet testaamaan kaikkia testitapauksia loppuun ennen testaukseen varatun ajan päättymistä.

Aineistosta poistettiin kaksi henkilöä, jotka vaikuttivat täysin laiminlyöneen testauksen; he myös lopettivat testauksen paljon etuajassa – käyttäen alle puolet (alle 60 minuuttia) testaukseen varatusta ajasta. Lisäksi aineistosta poistettiin yksi henkilö, joka ei nauhoituksen perusteella ollut ymmärtänyt laisinkaan testauksen tarkoitusta

(hän myös oli kirjoittanut tämän dokumentointiinsa) ja pääsi aloittamaan varsinaisen testauksen vasta testaukseen varatun ajan loppupuolella. Näin aineistosta poistettiin yhteensä kolme henkilöä. Näiden henkilöiden poistamisen jälkeen testaukseen, dokumentointiin ja muuhun testaukseen liittyvään toimintaan testaajat käyttivät keskimäärin 113 minuuttia testaukseen varatusta ajasta, joka oli 120 minuuttia (osallistujan testaukseen käyttämä aika määritettiin nauhoitusten perusteella). Lisäksi, henkilöiden poissulkemisen jälkeen ajankäytöllä ryhmien A (Md = 120 min) ja B (Md = 120 min) välillä ei ollut tilastollisesti merkitsevää eroa (p = 0,438, r = 0,105) (taulukko 3).

Taulukko 3. Yhteenveto ajankäytöstä ryhmien välillä sekä Wilcoxonin testin tulos ja sen korrelaatiokerroin r.

Muuttuja	Mediaani (keskipoikkeama)		Keskiarvo		*p*-arvo	w	*r*
Ajankäyttö	Ryhmä A $n = 26$	Ryhmä B $n = 30$	A	B	0,438	431,5	0,105
	120 (11,13)	120 (12,33)	114,6	111,5			

Samoin kuin ryhmien välillä, ohjelmien välisessä tarkastelussa ei ollut merkitseviä eroja virheiden lukumäärässä (p = 0,146, r = -0,138), vakavuudessa (p = 0,872, r = 0,015) ja F-arvossa (p = 0,393, r = 0,081) (taulukko 4). Tosin vääriä virheitä kirjattiin LibreOfficesta (ka = 1,23) enemmän kuin Trianglesta (ka = 0,61) (p = 0,001, r = -0,310) (taulukko 4).

Taulukko 4. Ohjelmien (Triangle ja LibreOffice) välisen eron Wilcoxonin järjestyssummatestin tulokset eri muuttujille.

Muuttuja	Testi-tapaukset	Mediaani (sd)	*p*-arvo	W	r
Virheiden lukumäärä	Triangle $n = 56$	3 (1,93)	0,146	1321	-0,138
	LibreOffice $n = 56$	4 (2,03)			
Vakavuus	Triangle $n = 56$	11 (6,02)	0,872	1595	0,015
	LibreOffice $n = 56$	10 (5,56)			
F-arvo	Triangle $n = 56$	0,06897 (0,301)	0,393	1567	0,081
	LibreOffice $n = 56$	0,05634 (0,277)			

Väärät virhe-merkinnät	Triangle $n = 56$	0 (0,76)	0,001	1037,5	-0,310
	LibreOffice $n = 56$	1 (1,08)			
Ulko-puolisuus	Triangle $n = 56$	0,6333 (0,5036)	0,011	1022,5	-0,241
	LibreOffice $n = 56$	1,143 (0,8430)			

6.2. F-arvo

F-arvo laskettiin molemmille ohjelmille erillisellä virheiden kokonaislukumäärällä, mutta kuten luvussa 5.4. Tarkkuus ja palautus nostettiin esille, tämä ei välttämättä ole täysin soveltuva tapa laskea F-arvo tämän kokeen yhteydessä, joten yhdistetty F-arvo laskettiin myös ja tätä myös päädyttiin lopulta käyttämään eritellyn sijasta (laskukaavassa on yhteinen kokonaisvirhelukumäärä molemmille ohjelmille). Eritellyllä F-arvolla ohjelmien välillä näyttäytyy selkeä ero johtuen LibreOfficessa olevasta huomattavasti suuremmasta virheiden lukumäärästä, mistä on havaittavissa erillisten virheiden kokonaislukumäärän käytön ongelma tässä kokeessa: Trianglen eritellyt F-arvot (Md = 0,312) ovat suuremmat kuin LibreOfficen (Md = 0,165) ($p < 0,001$, r = -0,485), mitä ei kuitenkaan yhdistetyllä F-arvolla ole johtuen yhteisestä virheiden kokonaislukumäärästä ($p = 0,393$, r = 0,081) (taulukko 5). Siitä huolimatta eritellyn tulos muutoin mukailee yhdistetyn tuloksia ryhmien sekä kevyiden ja raskaiden testitapausten vertailussa: ryhmien välillä sekä kevyiden ja raskaiden välillä on merkitsevä ero.

Taulukko 5. F-arvot ohjelmittain eritellyllä sekä yhdistetyllä F-arvon laskentakaavalla.

Muuttuja	Mediaani (keskipoikkeama)		Keskiarvo		*p*-arvo	W	r
F-arvo (yhd.)	Triangle $n = 56$	LibreOffice $n = 56$	Triangle	Libre-Office	0,393	1567	0,081
	0,06897 (0,301)	0,05634 (0,0277)	0,06366	0,0584			
F-arvo (eritelty)	0,300 (0,159)	0,165 (0,0755)	0,312	0,165	< 0,001	2395	0,485

6.3. Testitapausten ja virheiden suhde

Ennen tulosten käsittelyä saman virhemerkinnän monikerrat (duplikaattimerkinnät) poistettiin aineistosta. Duplikaattimerkinnät mukaan luettuna virhemerkintöjä tehtiin yhteensä 588 ja duplikaattimerkinnät poistettuna 523 kappaletta. Ohjelmista tehtiin kaiken kaikkiaan 99 virhemerkintää (oikeat ja väärät virhemerkinnät mukaan luettuna), joista 27 olivat Trianglessa ja loput 72 LibreOfficessa. Näistä virhemerkinnöistä 60 olivat todellisia virheitä ohjelmissa. Suurin virheiden lukumäärä, mitä yksittäinen henkilö kykeni löytämään, oli 16 ja keskimäärin testaajat löysivät kukin 7,5 virhettä (eli noin 12,5 % kaikista tunnetuista virheistä). Näin ohjelmissa olevien yksilöllisten oikeiden virheiden lukumäärä oli huomattavasti korkeampi kuin mitä kukaan yksittäinen henkilö kykeni löytämään ohjelmista. Keskimäärin testaajat löysivät 3,1 eksplisiittistä eli 51,5 % ohjelmissa olevista eksplisiittisistä virheistä, 2,2 implisiittistä (18,1 % kaikista implisiittisistä), 1,8 liittyvää ja 0,4 täysin ulkopuolista virhettä.

Raskailla testitapauksilla testaajat tekivät kaiken kaikkiaan 259 virhemerkintää löydetyistä virheistä, kun taas kevyillä tehtiin virheistä 169 virhemerkintää. Raskailla testitapauksilla löytyi enemmän virheitä kuin kevyillä ($p < 0{,}001$, $r = -0{,}387$) (taulukko 6). Kevyitä testitapauksia oli kuitenkin vähemmän johtuen niiden luonteesta (26 kevyttä ja 53 raskasta testitapausta), jolloin yhtä kevyttä testitapausta kohden löytyi keskimäärin 6,5 virhettä, kun raskailla testitapauksilla löytyi 4,73. Virheellisiä virhemerkintöjä tehtiin sekä raskailla, että kevyillä testitapauksilla, molemmilla keskimäärin noin 1. Kevyiden ja raskaiden testitapausten välillä ei ollut väärien virhemerkintöjen suhteen tilastollisesti merkitsevää eroa ($p = 0{,}294$, $r = -0{,}099$). Lisäksi, kokeen 79:tä testitapauksesta vain 11 testitapaukselle (14 %) ei kirjattu yhtään virhettä.

Taulukko 6. Kevyiden ja raskaiden testitapausten vaikutus löytyneisiin virheisiin.

Muuttuja	Testi-tapaukset	Mediaani (sd)	ka	*p*-arvo	W	r
Löytyneet virheet	Kevyt $n = 56$	3 (2,03)	3,02	< 0,001	873	-,387
	Raskas $n = 56$	5 (1,68)	4,48			

6.3.1. Kevyet ja raskaat testitapaukset

Molempien testitapausten joukkojen (kevyiden sekä raskaiden) tavoitteena oli kattaa samat ominaisuudet testattavista ohjelmista, mutta tämän toteuttaminen on vähintäänkin haasteellista kevyiden ja raskaiden testitapausten tarkkuuseron takia. Tämän lähtökohtaisen eron huomioimiseksi jokaiselle löytyneelle virheelle määritettiin ulkopuolisuuden taso suhteessa molempiin testitapausten joukkoihin. Tämä antaa osviittaa kunkin testitapauksen joukon tarkkuudesta virheiden kontekstissa.

Kaikkien kevyiden sekä raskaiden testitapausten Wilcoxonin järjestyssummatestien tulokset olivat seuraavat:

- raskailla testitapauksilla löytyi enemmän virheitä kuin kevyillä ($p < 0{,}001$, $r = -0{,}387$),
- raskaiden löytämät virheet olivat vakavampia ($p = 0{,}002$, $r = -0{,}295$),
- raskaat testitapaukset menestyivät kevyitä F-arvon suhteen paremmin ($p < 0{,}001$, $r = -0{,}290$),
- raskailla testitapauksilla löydetyt virheet olivat lähempänä testitapauksia ($p < 0{,}001$, $r = 0{,}652$) ja
- löytyneiden väärien virheiden lukumäärässä ei ollut eroa kevyillä ja raskailla testitapauksilla ($p = 0{,}294$, $r = -0{,}099$) (taulukko 7).

Taulukko 7. Molempien ohjelmien testitapausten rakenteen (kevyet ja raskaat testitapaukset) Wilcoxonin järjestyssummatestin tulokset.

Muuttuja	Testi-tapaukset	Mediaani (sd)	*p*-arvo	W	r
Virheiden lukumäärä	Kevyt $n = 56$	3 (2,03)	< 0,001	873	-0,387
	Raskas $n = 56$	5 (1,68)			
Vakavuus	Kevyt $n = 56$	8,5 (5,85)	0,002	1033	-0,295
	Raskas $n = 56$	13,5 (5,21)			
F-arvo	Kevyt $n = 56$	0,0423 (0,0284)	< 0,001	879,5	-0,325
	Raskas $n = 56$	0,0704 (0,0267)			
Väärät virhe-merkinnät	Kevyt $n = 56$	1 (0,94)	0,294	1398	-0,099
	Raskas $n = 56$	1 (1,02)			
Ulkopuolisuus	Kevyt $n = 56$	1,333 (0,673)	< 0,001	2531,5	0,652
	Raskas $n = 56$	0,400 (0,440)			

Samat testit tehtiin myös ohjelmakohtaisesti kevyille ja raskaille näiden mahdollisen eron tarkastelemiseksi. Trianglen kohdalla tulokset olivat:

- Trianglen raskaat testitapaukset menestyivät F-arvon suhteen kevyitä paremmin (p = 0,003, W = 175, r = -0,405),
- virheitä löytyi raskailla testitapauksilla enemmän Trianglella (p < 0,001, W = 175, r = -0,478),
- virheet olivat vakavampia raskailla testitapauksilla Trianglella (p = 0,001, W = 194, r = -0,431),
- Trianglesta löytyneiden virheiden etäisyys testitapauksista oli raskailla testitapauksilla pienempi (p < 0,001, W = 571, r = 0,573) ja
- väärien virheiden lukumäärässä ei ollut merkitsevää eroa (p = 0,927, W = 384,5, r = 0,014) (taulukko 8).

Taulukko 8. Trianglen Wilcoxonin järjestyssummatestien tulokset kevyille ja raskaille testitapauksille.

Muuttuja	Testi-tapaukset	Mediaani (sd)	*p*-arvo	W	r
Virheiden lukumäärä	Kevyt $n = 30$	2,5 (1,81)	< 0,001	175	-0,478
	Raskas $n = 26$	4 (1,58)			
Vakavuus	Kevyt $n = 30$	7,5 (5,84)	0,001	194	-0,431
	Raskas $n = 26$	14,5 (5,09)			
F-arvo	Kevyt $n = 30$	0,052 (0,0278)	0,003	175	-0,405
	Raskas $n = 26$	0,069 (0,0272)			
Väärät virhe-merkinnät	Kevyt $n = 30$	0,633 (0,850)	0,927	384,5	-0,014
	Raskas $n = 26$	0,577 (0,643)			
Ulkopuolisuus	Kevyt $n = 30$	1,0 (0,541)	< 0,001	571	0,573
	Raskas $n = 26$	0,4 (0,280)			

LibreOfficella puolestaan:

- raskailla testitapauksilla löytyi kevyitä testitapauksia enemmän virheitä (p = 0,045, W = 269,5, r = -0,269),

- vakavuudessa ei ollut eroa (p = 0,242, W = 318,5, r = -0,157),
- F-arvon suhteen ei ollut eroa (p = 0,070, W = 269,5, r = -0,243).
- virheiden ulkopuolisuuden taso oli kevyillä testitapauksilla raskaita lähempänä testitapauksia (p < 0,001, W = 719, r = 0,782) ja
- väärien virheiden lukumäärässä ei ollut merkitsevää eroa (p = 0,215, W = 317, r = 0,167) (taulukko 9).

Taulukko 9. LibreOfficen Wilcoxonin järjestyssummatestien tulokset kevyille ja raskaille testitapauksille.

Muuttuja	Testi-tapaukset	Mediaani (sd)	*p*-arvo	W	r
Virheiden lukumäärä	Kevyt $n = 26$	3 (2,18)	0,045	269,5	-0,269
	Raskas $n = 30$	5 (1,80)			
Vakavuus	Kevyt $n = 26$	9 (5,90)	0,242	318,5	-0,157
	Raskas $n = 30$	12,5 (5,22)			
F-arvo	Kevyt $n = 26$	0,042 (0,0295)	0,071	269,5	-0,243
	Raskas $n = 30$	0,070 (0,0253)			
Väärät virhe-merkinnät	Kevyt $n = 26$	1,039 (0,999)	0,215	317	-0,167
	Raskas $n = 30$	1,400 (1,133)			
Ulkopuolisuus	Kevyt $n = 26$	2,00 (0,396)	< 0,001	719	0,782
	Raskas $n = 30$	0,414 (0,543)			

6.3.2. *Testitapausten testikattavuuden ulkopuolisen alueen määritys*

Testitapauksen ulkopuolisen alueen tarkastelemiseksi virhemerkinnöille määrättiin neljä ulkopuolisuuden tasoa suhteessa testitapauksiin (tasojen numerointi vastaa ordinaalista järjestysasteikkoa):

- eksplisiittiset (0),
- implisiittiset (1),

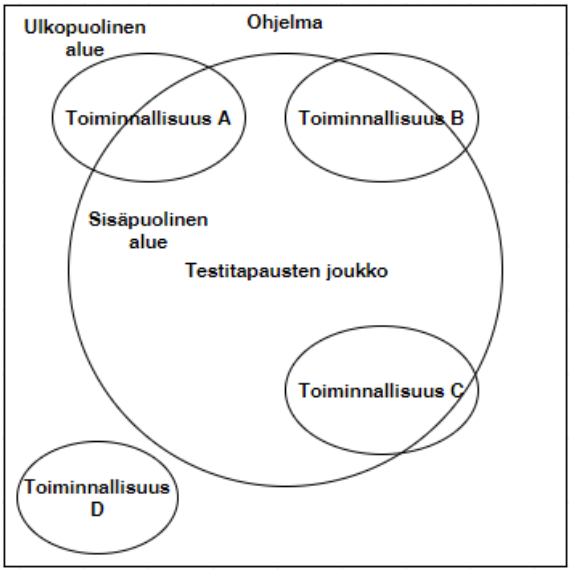
- liittyvät (2) ja
- täysin ulkopuoliset (3).

Kuvissa 7 ja 8 on esitetty käsitekarttojen muodossa testitapausjoukon ja sen määräämät ohjelman alueet (eli testitapausten sisä- ja ulkopuolinen alue) suhteessa testitapausjoukkoon (eksplisiittinen, implisiittinen, liittyvä ja täysin ulkopuolinen alue). Kuva 7 on havainnollistus ohjelmasta, joka koostuu neljästä toiminallisuudesta (A, B, C ja D), joille on laadittu testitapausten joukko. Tällä testitapausten joukolla saadaan katettua ohjelmasta alue, jonka sisäpuolelle saadaan sisällytettyä osa ohjelmasta, mutta osa ohjelman toiminnallisuuksista jää testitapausten ulkopuolelle. Sama neljän toiminnallisuuden ohjelma ja sille määrätty testitapausjoukko on esitettynä kuvassa 8 yksityiskohtaisemmin eri ulkopuolisuuksien tasojen suhteen, jotka sisältyvät sisä- ja ulkopuoliseen alueeseen. Virheet, jotka löytyvät joltain näiltä alueelta voidaan määritellä samalla periaatteella kyseisten tasojen virheiksi. Tässä neljän tason jaottelussa,

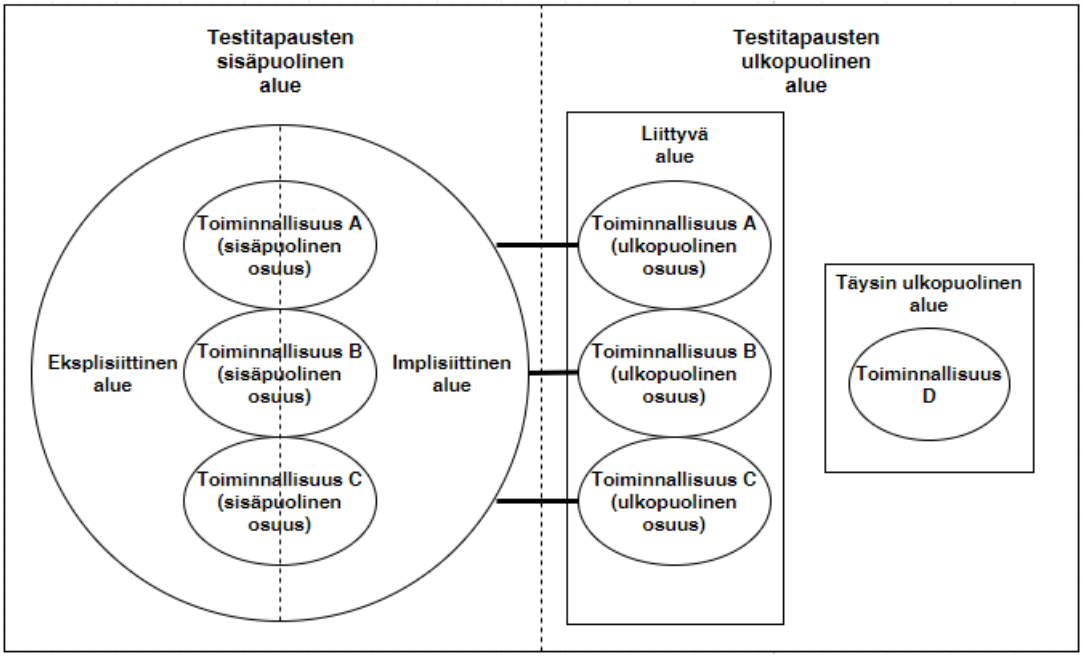
- eksplisiittisille virhemerkinnöille voidaan vetää *suora yhteys* testitapauksen eksplisiittiseen sisältöön,
- implisiittisille virhemerkinnöille voidaan vetää *epäsuora yhteys* testitapausten sisältöön (ts. testitapaus ei välttämättä ota suoraa kantaan sisältöön, mutta käytännössä se on kuitenkin sisältyneenä testitapaukseen),
- liittyville virhemerkinnöille *ei voida vetää yhteyttä* testitapauksiin, mutta virhemerkinnän sisältö kuitenkin liittyy testitapaukseen, kun taas
- täysin ulkopuolisille virheille ei voida vetää *ollenkaan yhteyttä* testitapauksiin.

Tässä luokittelussa eksplisiittiset ja implisiittiset (0 ja 1) virhemerkinnät katsottiin olevan ns. testitapausten sisäpuolisia virheitä, eli nämä virheet ovat testitapausten kattamia. Jos taas virhemerkintä oli liittyvä tai täysin ulkopuolinen (2 tai 3); se katsottiin testitapausten ulkopuoliseksi virheeksi, eli nämä virheet eivät olleet testitapausten kattamia.

Taulukossa 10 on esitettynä jokaiselta neljältä ulkopuolisuuksien tasolta virheet, jotka löytyivät testatuista ohjelmista. Liitteessä 4. Ohjelmista löytyneet virheet ja niiden luokitukset on vielä esitettynä kaikki ohjelmista löytyneet virheet, niiden vakavuusluokitukset, ulkopuolisuuden tasot suhteessa molempiin testitapausten joukkouhin, ohjelma, josta virheet löytyivät, sekä kuinka moni testaaja löysi virheet.



Kuva 7. Esimerkki testitapausten joukon rajoittamasta alueesta ohjelman kokonaisuudesta, joka määrittää testitapausten ulko- ja sisäpuolisen alueen ohjelmasta sekä ohjelman toiminnallisuuksien suhde testitapausten joukkoon.



Kuva 8. Testitapausten määräämät ulkopuolisuuden tasot: eksplisiittinen, implisiittinen, liittyvä sekä täysin ulkopuolinen.

Taulukko 10. Esimerkit virheistä kaikista neljästä eri ulkopuolisuuksien tasosta, jotka löytyivät testatuista ohjelmista.

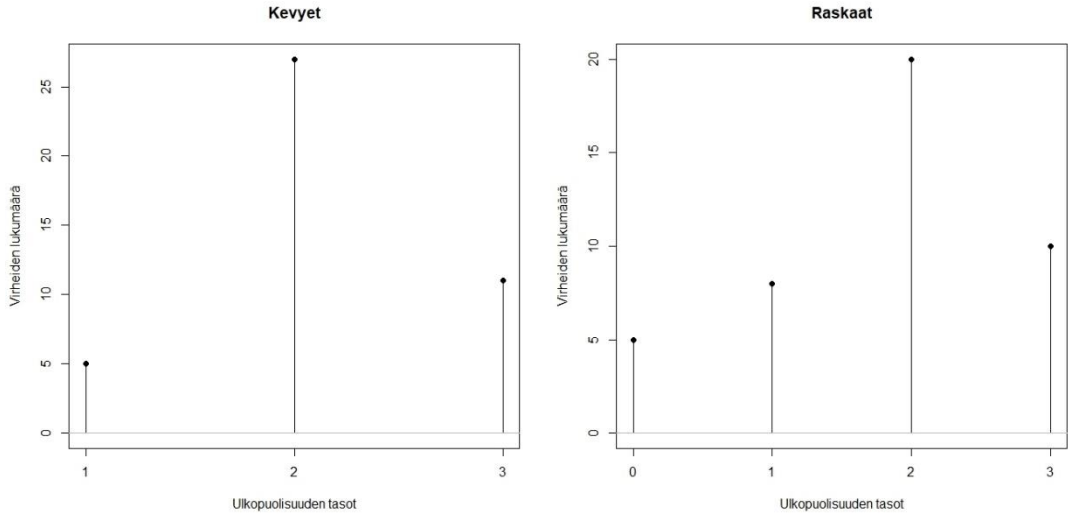
<table>
<tr><th>Testitapauksen kuvaus</th><th>Virheen kuvaus</th><th>Ulkopuolisuus</th><th>Perustelu</th></tr>
<tr><td>Verify that only positive values are accepted for triangle sides</td><td>Negatiiviset arvot hyväksytään syötekenttään, vaikka sen ei pitäisi olla mahdollista.</td><td>Eksplisiittinen</td><td>Virheen löytämiseksi tehtävä testaus on selkeästi esitetty testitapauksissa.</td></tr>
<tr><td>Test search field</td><td rowspan="2">Pikanäppäimiä ei voi käyttää search-valikon ollessa auki.</td><td rowspan="2">Implisiittinen</td><td rowspan="2">Pikanäppäinten ja hakuvalikon yhdistävää testitapausta ei ollut, mutta pikanäppäinten testit olivat seuraavat testit haun testien jälkeen, jolloin iso osa testaajista (15 testaajaa) huomasi ja dokumentoi kyseisen virheen. Näin virheen testaamiseksi ei ollut ohjetta, mutta virhe tuli sisällytetyksi testeihin kuitenkin.</td></tr>
<tr><td>Test all hotkeys for special character page</td></tr>
<tr><td>Test insert using right mouse button</td><td>Arabialaisten kirjainten syöttö ei toimi aina oikein: välillä teksti syötetään väärälle puolelle latinalaisten kirjainten kanssa. (Arabiaa kirjoitetaan oikealta vasemmalle)</td><td>Liittyvä</td><td>Arabialaisten kirjainten syöttämistä varten ei ollut testitapauksia eikä myöskään kirjainten syöttämissuuntaan liittyen. Kuitenkin useat testit käsittelivät kirjainten syöttämistä tekstiin ja siten kirjainten syöttäminen liittyi testitapauksiin.</td></tr>
<tr><td>-</td><td>Oikealle painaminen näppäimistöstä liikuttaa kursoria vasemmalle arabialaisilla kirjaimilla.</td><td>Täysin ulkopuolinen</td><td>Mitään tämän kaltaista ei ollut testitapauksissa.</td></tr>
</table>

Eksplisiittisen tason virhemerkinnät ovat suoraan löydettävissä toteuttamalla testitapauksen kuvaus ja näin ne ovat virheitä, jotka sisältyvät testitapauksen kuvaukseen. *Eksplisiittiset virhemerkinnät* ovat siten löydettävissä testitapauksen osien; otsikon, kuvauksen, annettujen toimenpiteiden, oletettujen tulosten tai muiden sen osien pohjalta. Tällöin virhemerkinnän etäisyyttä testitapauksista kuvataan arvolla

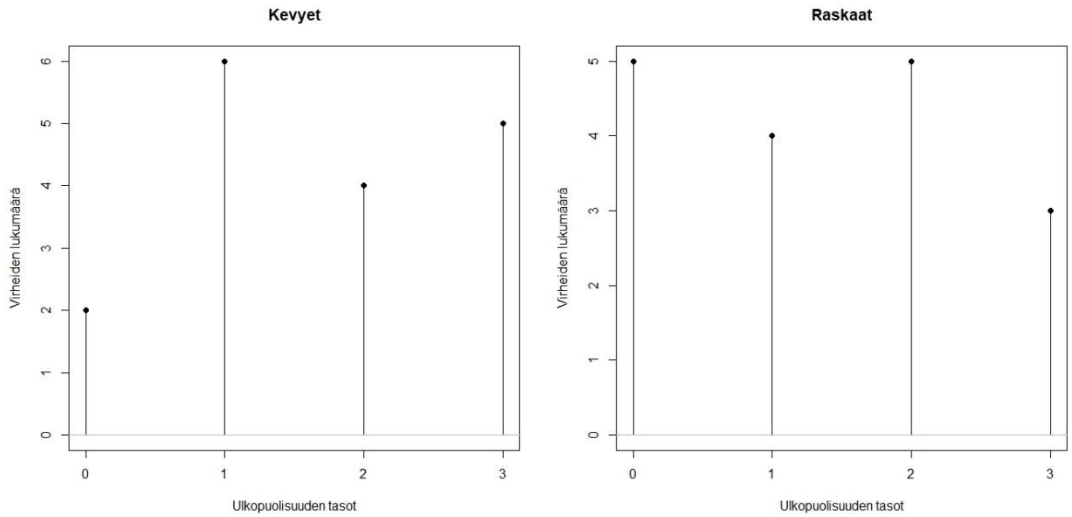
0. Jos taas virhe löytyi ohjelman alueesta, joka tulee testatuksi testattaessa testitapauksilla, mutta testitapaus ei eksplisiittisesti ota kantaa kyseisen alueen testaamiseen; on virhe implisiittinen (ja etäisyys on silloin 1). *Implisiittiset virhemerkinnät* taas löytyvät testitapausta testaamalla vaikka niihin liittyvään toimintaan ei otettaisi kantaa testitapauksen sisällössä, ts. ne ovat testitapauksen sanomatonta sisältöä. Juuri sen tähden ne ovat myös luettu testitapausten sisäpuolisiin virhemerkintöihin. Etäisyyden arvo 2 viittaa ns. *liittyviin virhemerkintöihin*, joiden ominaisuuksia tai toiminnallisuuksia testitapaukset käsittelivät jollain tapaa, mutta testitapausten sisällön noudattaminen ei itsessään riittäisi merkinnän käsittelemän virheen löytämiseksi. *Liittyvät virhemerkinnät* siis nimensä mukaisesti liittyvät testitapauksissa käsiteltyihin asioihin (esim. virhe sisältyy toiminnallisuuteen, jota on käsitelty testitapauksella), mutta testitapaukset ovat riittämättömiä kyseisten virheiden löytämiseksi. *Täysin ulkopuolisilla virhemerkinnöillä* (etäisyyden arvolla 3) tarkoitetaan kaikista kauimpia virheitä testitapauksista: ne ovat virhemerkintöjä, joiden ei voi katsoa liittyvän ollenkaan käsiteltyihin testitapauksiin ja, joita ei voi katsoa löytyvän pelkästään testaamalla testitapausten käsittelemiä asioita. *Täysin ulkopuoliset virhemerkinnät* ovat siis täysin testitapausten ulkopuolella, jolloin niitä ei voi suoraan tai epäsuorasti johtaa testitapauksista, eikä testitapausten perusteella voisi päätellä kyseisiä virheitä testitapausten avulla löytyvän.

6.3.3. Havainnot ulkopuolisuuksien tasoilta

Kuva 9 ja 10 esittävät kevyiden ja raskaiden testitapausten ulkopuolisuus profiilit LibreOfficella sekä Trianglella. Kuvissa on esitetty kaikki ohjelmien virheet ja niiden suhde sekä kevyisiin että raskaisiin testitapauksiin. Näiden virheiden on oletettu kattavan riittävällä tarkkuudella testattujen ohjelmien osien kaikki virheet (perustelut tälle on esitetty tarkemmin luvussa 5.4. Tarkkuus ja palautus).



Kuva 9. LibreOfficen testattujen kokonaisuuksien virheiden jakautuminen ulkopuolisuuksien tasoille (ulkopuolisuus profiili), jossa 0 = eksplisiittinen, 1 = implisiittinen, 2 = liittyvä ja 3 = täysin ulkopuolinen.



Kuva 10. Trianglen virheiden jakautuminen ulkopuolisuuksien tasoille (ulkopuolisuus profiili), jossa 0 = eksplisiittinen, 1 = implisiittinen, 2 = liittyvä ja 3 = täysin ulkopuolinen.

Näistä kuvista on havaittavissa eroavaisuuksia – ei vain virheiden jakautumisesta ohjelmien kesken, mutta myös ohjelmille käytetyistä testitapauksista. Erityisesti LibreOfficessa olevat virheet olivat painottuneempia testitapausten ulkopuolelle, Trianglen painotus ei ollut läheskään näin selkeä. Kuvaajista voi myös havaita Trianglen kevyiden testitapausten sisältävän suhteellisesti enemmän eksplisiittisiä ja implisiittisiä virheitä kuin LibreOfficen raskaat testitapaukset. Huomion arvoista on myös, että LibreOfficen kevyillä testitapauksilla ei ollut löydettävissä yhtään eksplisiittistä virhettä. Trianglesta löytyneet virheet olivatkin painottuneempia eksplisiittisiin ja implisiittisiin virheisiin, kuin LibreOfficessa: Trianglen (ka = 0,71) ja LibreOfficen (ka = 1,18) välillä keskimääräisessä ulkopuolisuuden tasossa on tilastollisesti merkitsevä ero ($p = 0{,}011$, $r = -0{,}241$). Kevyet ja raskaat testitapaukset poikkesivat toisistaan eksplisiittisten ($p < 0{,}001$, $W = 200{,}5$, $r = -0{,}779$), liittyvien ($p = 0{,}009$, $W = 1968$, $r = 0{,}248$) ja täysin ulkopuolisten virheiden ($p = 0{,}030$, $W = 1820$, $r = 0{,}205$) lukumäärässä, muttei implisiittisten ($p = 0{,}338$, $W = 1410{,}5$, $r = -0{,}091$) (taulukko 11).

Keskimääräisestä yhden henkilön löytämästä 7,5 virheestä oli 3,1 eksplisiittisiä, 2,2 implisiittisiä, 1,8 liittyvää ja 0,4 täysin ulkopuolista virhettä. Testaajat löysivät siis keskimäärin 51,5 % löydettävissä olevista eksplisiittisistä, 18,1 % implisiittisistä, 6,5 % liittyvistä ja 2,6 % ulkopuolisista virheistä.

Taulukko 11. Virheiden ulkopuolisuuden tasot kevyille ja raskaille testitapauksille.

Muuttuja	Testi-tapaukset	Keskiarvo (sd)	Md	*p*-arvo	W	r
Eksplisiittiset	Kevyt $n = 56$	0,375 (0,620)	0	< 0,001	200,5	-0,779
	Raskas $n = 56$	2,750 (1,240)	3			
Implisiittiset	Kevyt $n = 56$	1,02 (0,981)	1	0,338	1410,5	-0,091
	Raskas $n = 56$	1,18 (0,917)	1			
Liittyvät	Kevyt $n = 56$	1,357 (1,84)	0,5	0,009	1968	0,248
	Raskas $n = 56$	0,446 (0,761)	0			
Täysin ulkopuoliset	Kevyt $n = 56$	0,268 (0,447)	0	0,030	1820	0,205
	Raskas $n = 56$	0,107 (0,312)	0			

6.4. Kvalitatiiviset tulokset

Nauhoituksista kävi ilmi, että osa osallistujista kirjasi virheitä sopivamman testitapauksen kohdalle ja toiset taas kirjasivat sen testitapauksen kohdalle, jolloin he löysivät virheen. Epäselvien virhemerkintöjen virheiden löytymisajankohtien nauhoituksia katsottaessa selvisi; oliko löydös oikea ja oliko virhemerkintä todellinen virhe. Mikäli virhemerkintää ei pystynyt toistamaan ja nauhoituksen eikä dokumentaation perusteella voinut todentaa merkintää virheeksi, virhemerkintää ei kelpuutettu oikeaksi virheeksi.

Virheiden dokumentoinnin taso ei ollut kovin hyvä monella osallistujalla: pelkkä dokumentoinnin teksti ei aina riittänyt ymmärtämään virhemerkinnässä esitettyä virhettä. Ilman visuaalista informaatiota (eli osallistujien ottamat kuvat ja testauksen nauhoitukset) monet virhemerkinnät olisivat jääneet epäselviksi ja niissä esitetyt virheet olisivat jääneet löytymättä. Visuaalinen informaatio myös nopeutti selkeästi virheiden ymmärtämistä, jolloin virhemerkintä oli nopeammin tulkittavissa ja helpompi toistaa.

Testauksen nauhoitus toi myös ilmi, että vaikka virhe olisi kirjattu jonkin tietyn testitapauksen kohdalle, se ei suoraan tarkoittanut, että kyseinen virhe olisi löytynyt kyseisellä testitapauksella. Nauhoituksessa joidenkin virheiden dokumentoinnin aikana oli havaittavissa, että testaaja etsi sopivaa testitapausta löytämälleen virheelle ja päätyi lopulta kirjaamaan virheen eri testitapauksen kohdalle, kuin minkä testitapauksen aikana hän löysi virheen.

7. POHDINTA

Tämän opinnäytetyön yhteydessä tehdyn kokeen päämääränä oli vastata tutkimuskysymyksiin:

- ”*Kuinka suuri osa käsikirjoitetun testauksen löytämistä ohjelmistovirheistä ovat käsikirjoitettujen testitapausten kattamia?”* sekä
- ”*Vaikuttaako testitapausten testausohjeistuksen rakenne ulkopuolisten virheiden esiintymiseen?”*

Näihin vastaamiseksi määritettiin ulkopuolisuuden tasot; eksplisiittinen, implisiittinen, liittyvä sekä täysin ulkopuolinen, jotka kuvaavat testitapausten ja sen määräämän alueen ohjelmasta ja siten myös virheiden sijoittumisen näille tasoille. Näiden avulla voidaan tarkemmin tarkastella virheiden sijoittumista testitapausten kattamalle alueelle (testitapausten sisäpuolelle) sekä kattavuuden ulkopuolelle (testitapausten ulkopuolelle).

Kokeen tulokset viittaavat merkittävän osuuden ohjelmien virheistä löytyvän testitapausten kattamalta alueelta, mutta siitä huolimatta – testitapausten ulkopuolelta löytyy paljon virheitä. Testaajat vaikuttavat löytävän testitapauksia noudattamalla myös niiden ulkopuolelta virheitä, mutta niiden löytyminen vaikuttaa epätodennäköisemmältä kuin testitapausten sisäpuolisten virheiden löytyminen. Myöskin testitapausten sisäpuolisten virheiden löytyminen ei ole itsestään selvyys, vaikka testitapausten testiohjeiden eksplisiittinen noudattaminen pitäisi johtaa virheen löytämiseen – merkittävä osuus testitapausten sisäpuolisista (eksplisiittistä sekä implisiittisistä) virheistä jäi löytymättä. Testitapausten testausohjeistuksen rakenteen eroavaisuuksia ei kyetty tarkastelemaan, sillä virheen ulkopuolisuus määritetään sen suhteesta testitapauksiin, ja testitapausten rakenteen ollessa eri; on myös ulkopuolisuus eri. Muutoin testauksesta kerätyn aineiston perusteella testitapausten rakenteen vaikutusta kyettiin vertaamaan ja niiden välillä oli selkeä tilastollinen ero (käyttäen Wilcoxonin merkittyjen sijalukujen testiä): yksityiskohtaisemmat (eli raskaammat) testitapaukset menestyivät karkeampia (eli kevyempiä) testitapauksia paremmin.

Toteutetun koeasetelman ollessa uusi sekä jo kokeen suunnittelun yhteydessä asetelmassa tunnettiin olevan rajoituksensa (5.5. Rajoitukset), koeasetelmalle selvitettiin ohjelmien ja ryhmien mahdollinen vaikutus Wilcoxonin testillä (testi käsitelty luvun 6. Tulokset alussa). Tämän vertailun perusteella katsottiin olevan perusteltua tarkastella testitapausten rakenteen vaikutusta testaustuloksiin, jota koeasetelmalla pyrittiin tarkastelemaan.

7.1. Koeasetelman vaikutus

Ohjelmissa oleva todellinen virheiden kokonaislukumäärä voidaan olettaa olevan riittävän lähellä löytyneiden virheiden lukumäärää tässä koeasetelmassa, sillä ne pitivät sisällään kaikki ammattitestaajien testauksen löytämät virheet sekä virheiden lukumääräksi tuli vieläpä huomattavasti suurempi määrä kuin kukaan yksittäinen henkilö kykeni löytämään. Tehdyistä toimenpiteistä huolimatta testatuissa ohjelmissa,

erityisesti LibreOfficessa voi olla erityisosaamista tai tietoa vaativaa (esim. tietoturva tai suorituskyky) tai muutoin hyvin vaikeasti löydettävissä olevia virheitä, mitä kokonaisvirheiden lukumäärä ei pidä sisällään.

Testitapausten jakamisella kahteen osaan (A ja B) pyrittiin kontrolloimaan yksittäisen ohjelman vaikutusta kokeessa käytettyihin ohjelmiin sekä varmistua ylipäätään testaajien kyvystä suoriutua testauksesta. Wilcoxonin testit tehtiin myös ryhmille sekä pelkästään ohjelmille koeasetelman tarkastelemiseksi. Tulokset viittaavat siihen, että molemmissa näissä onnistuttiin. Ryhmien A ja B välillä ei ole tilastollisesti merkitsevää eroa virheiden lukumäärän ($p = 0{,}366$, $r = 0{,}122$), vakavuuden ($p = 0{,}167$, $r = 0{,}186$), virheellisten virhemerkintöjen lukumäärän ($p = 0{,}251$, $r = -0{,}154$) tai F-arvojen suhteen ($p = 0{,}218$, $r = 0{,}166$), mikä on positiivinen merkki ryhmiin jaottelun onnistumisesta. Virheiden etäisyydessä oli kuitenkin selkeä ero A:n ja B:n välillä: A-ryhmän löytämien virheiden etäisyys oli suurempi ($p < 0{,}001$, $r = 0{,}524$) (A-ryhmä testasi Trianglen raskailla testitapauksilla ja LibreOfficen kevyillä). Trianglen virheet (ka = 0,71) olivat lähempänä testitapauksia LibreOfficeen (ka = 1,18) verrattuna ($p = 0{,}011$, $r = -0{,}241$) ja LibreOfficen kevyiden testitapausten etäisyys raskaista oli erityisen suuri ($p < 0{,}001$ ja $r = 0{,}782$), joka oli Trianglen kevyiden ja raskaiden testitapausten eroa suurempi ($p < 0{,}001$, $r = 0{,}573$). Se kertoo, että kaikki ryhmien välillä kevyet ja raskaat testitapaukset eivät olleet aivan yhtäläisiä. Tästä etäisyys erosta huolimatta kokeessa käytetyllä ryhmiin jaottelulla itsellään ei näyttäisi kaiken kaikkiaan olevan suurta vaikutusta kokeen tuloksiin, mikä olikin sen tavoitteena.

Ohjelmista (Triangle ja LibreOffice) löydettyjen virheiden lukumäärässä ($p = 0{,}146$, $r = -0{,}138$), vakavuudessa ($p = 0{,}872$, $r = 0{,}015$) ja F-arvossa ($p = 0{,}393$, $r = 0{,}081$) ei ollut eroa ohjelmien välillä. LibreOfficesta kirjattiin enemmän vääriä virheitä (ka = 1,23) kuin Trianglesta (ka = 0,61) ($p = 0{,}001$, $r = -0{,}310$). Täten ryhmiin jaottelulla (etäisyys ero) ja ohjelmilla (väärien virheiden lukumäärä) on osoitettavissa oleva vaikutus (joskin ei suuri) kokeen tuloksiin, mikä korostaa kokeen tulosten yleistettävyyden ongelmallisuutta. Monissa testaukseen liittyvässä tutkimuksissa onkin usein mainittu rajoitteeksi niissä käytettyjen ohjelmien yleistettävyys, esimerkiksi Camilo F. 2015 ja Mäntylä M. ja Itkonen J. 2013 [41, 69]. Selkeästi sama ongelma on myös tässä kokeessa. Tästä rajoituksesta huolimatta kokeen tulokset ovat lupaavia.

Vaikkakin 20% parhaista suorituksista palkittiin, tämä ei ollut riittävä saamaan aivan kaikkia osallistujia käyttämään koko kokeeseen varattua aikaa testaamiseen. Kolme suoritusta jouduttiin sulkemaan tuloksista pois ja kaksi näistä poistettiin juuri raa'asti lyhyen (ajankäyttö alle puolet kokeeseen varatusta ajasta) ja laiminlyödyn suorituksen vuoksi. Ajankäytöllisesti ryhmien välillä ei ollut tilastollista eroa kelpuutettujen suoritusten osalta, mutta kaikki nämä kolme henkilöä olivat A-ryhmästä (eli raskaat Trianglen testitapaukset ja kevyet LibreOfficen testitapaukset). Tässä ryhmässä oli pienempi lukumäärä testitapauksia, mikä on saattanut vaikuttaa suoritusten riittämättömyyteen. Lisäksi, raskaat testitapaukset olivat tässä ryhmässä kohdennettu yksinkertaisempaan ohjelmaan (Triangle), millä on voinut myös olla oma vaikutuksensa. Nauhoituksen perusteella vaikutti monen kokevan olevan valmis suorituksessaan testattuaan testitapaukset kerran lävitse. Testitapausten testaaminen loppuun saattaa tuottaa kokemusta, että testattu ominaisuus on testattu riittävän hyvin.

7.2. Virheiden ja testitapausten suhde: ulkopuolisuuden tasot

Alustavaa jako pelkästään ohjelmistovirheiden sijoittumisesta testitapausten testikattavuuden sisä- ja ulkopuoliselle alueelle nähtiin riittämättömänä tarkastelemaan tätä ilmiötä. Jotta asioita olisi helpompi käsitellä; niille on hyvä antaa nimi. Siispä kokeen edetessä päädyttiin luomaan tarkempi tapa tarkastella ulkopuolisuuden tasoja nelitasoisella jaottelulla:

- eksplisiittinen (0),
- implisiittinen (1),
- liittyvä (2) sekä
- täysin ulkopuolinen alue (3).

Kuvat 7 ja 8 havainnollistavat näitä ulkopuolisuuksien tasoja ja niiden suhdetta testattavaan ohjelmaan. Nämä neljä ulkopuolisuuden tasoa luotiin uutena metriikkana kokeen yhteydessä auttamaan vastaamaan molempiin tutkimuskysymykseen, jotka olivat muotoa: ”*Kuinka suuri osa käsikirjoitetun testauksen löytämistä ohjelmistovirheistä on käsikirjoitettujen testitapausten kattamia?”* sekä ”*Vaikuttaako testitapausten testausohjeistuksen rakenne ulkopuolisten virheiden esiintymiseen?”*

Eksplisiittisen tason virhemerkinnät ovat suoraan löydettävissä toteuttamalla testitapauksen kuvaus ja näin ne ovat virheitä, jotka sisältyvät testitapauksen kuvaukseen. *Eksplisiittiset virhemerkinnät* ovat siten löydettävissä testitapauksen osien; otsikon, kuvauksen, annettujen toimenpiteiden, oletettujen tulosten tai muiden sen osien pohjalta. Tällöin virhemerkinnän etäisyyttä testitapauksista kuvataan arvolla 0. Jos taas virhe löytyi ohjelman alueesta, joka tulee testatuksi testattaessa testitapauksilla, mutta testitapaus ei eksplisiittisesti ota kantaa kyseisen alueen testaamiseen; on virhe implisiittinen (ja etäisyys on silloin 1). *Implisiittiset virhemerkinnät* taas löytyvät testitapausta testaamalla vaikka niihin liittyvään toimintaan ei otettaisi kantaa testitapauksen sisällössä, ts. ne ovat testitapauksen sanomatonta sisältöä. Juuri sen tähden ne ovat myös luettu testitapausten sisäpuolisiin virhemerkintöihin. Etäisyyden arvo 2 viittaa ns. *liittyviin virhemerkintöihin*, joiden ominaisuuksia tai toiminnallisuuksia testitapaukset käsittelivät jollain tapaa, mutta testitapausten sisällön noudattaminen ei itsessään riittäisi merkinnän käsittelemän virheen löytämiseksi. *Liittyvät virhemerkinnät* siis nimensä mukaisesti liittyvät testitapauksissa käsiteltyihin asioihin (esim. virhe sisältyy toiminnallisuuteen, jota on käsitelty testitapauksella), mutta testitapaukset ovat riittämättömiä kyseisten virheiden löytämiseksi. *Täysin ulkopuolisilla virhemerkinnöillä* (etäisyyden arvolla 3) tarkoitetaan kaikista kauimpia virheitä testitapauksista: ne ovat virhemerkintöjä, joiden ei voi katsoa liittyvän ollenkaan käsiteltyihin testitapauksiin ja, joita ei voi katsoa löytyvän pelkästään testaamalla testitapausten käsittelemiä asioita. *Täysin ulkopuoliset virhemerkinnät* ovat siis täysin testitapausten ulkopuolella, jolloin niitä ei voi suoraan tai epäsuorasti johtaa testitapauksista, eikä testitapausten perusteella voisi päätellä kyseisiä virheitä testitapausten avulla löytyvän.

Ulkopuolisuuden tasot (voisi kutsua myös virhe-etäisyydeksi) on järjestysasteikko, eikä sitä voida pitää absoluuttisena etäisyyden asteikkona. Asteikon arvojen rajat eivät myöskään ole täysin ehdottomat: ne ovat jossain määrin tulkinnanvaraisia. Testitapausten rakenteella on huomattava vaikutus näiden ulkopuolisuuden tasojen

kokoon. Ulkopuolisuuden voisi myös määrittää ainakin kahdella eri tavalla löytyneille virheille: se voitaisiin määrittää testitapausten ja virheiden suhteesta tai jonkun muun testikattavuuden yksikön ja virheen suhteesta (jälkimmäistä ei ole yritetty soveltaa laisinkaan). Lisäksi, testitapausten ja virheiden ulkopuolisuuden voisi myös määrittää ainakin kahdella eri tavalla: joko ulkopuolisuus suhteessa koko testitapausten joukkoon tai ulkopuolisuus suhteessa löytyneelle virheelle ja testitapaukselle, jolla virhe kirjattiin löytyneeksi. Tässä aineistossa käytettiin ulkopuolisuutta suhteessa koko testitapausten joukkoon, sillä ulkopuolisuus määrättiin *uniikeille virhemerkinnöille* eikä jokaiselle yksittäiselle virhemerkinnälle. Jälkimmäisen vaihtoehdon käyttäminen katsottiin soveltumattomaksi.

Vaikka ulkopuolisuuden kuvaamiseksi käytettiin nelitasoista jaottelua, jaon kuitenkin pystyisi tekemään useammallakin eri tavalla: rajat tasojen välillä ovat jossain määrin häilyvät sekä voi olla mahdollista, että jaottelun voisi tehdä yksityiskohtaisemmin. Osa *implisiittisistä virheistä* voisi katsoa olevan eksplisiittisesti johdettavissa testitapauksista, jolloin ne olisi löydettävissä testitapauksen sanomattoman (oletetun tai olettamattoman) sisällön perusteella. *Implisiittiset virheet* voisi myös katsoa olevan ulkopuolisia virheitä, sillä niitä ei ole eksplisiittisesti määritelty testitapauksissa. Toisaalta taas *ulkopuolistenkin virheiden* voisi katsoa olevan *implisiittisiä virheitä*, vaikka niitä ei voisi olettaa löytyvän pelkästään tarkastelemalla testitapauksia; ne on kuitenkin löydetty käyttäen kyseisiä testitapauksia testauksen perustana. Käytetyssä luokittelussa liittyvät virheet voisi katsoa olevan osittain testitapausten ansioita, jolloin ne voisi myös mahdollisesti luokitella sisäpuolisiin virheisiin. Tähän vaikuttaa oleellisesti se, että oletetaanko testaajan testaavan muutakin kuin vain testitapausten eksplisiittinen sisältö.

Ulkopuolisuuden tasoja voisi myös ajatella “aiotun testauksen sisällön” ja “aikomattoman testauksen sisällön kautta”. Ulkopuolisuuksien tasojen näkökulmasta liittyvät virheet ovat usein sellaisia, että ne on ollut tarkoitus löytää testitapauksilla. Kun taas täysin ulkopuoleisilta virheiltä on jäänyt puuttumaan aikomus kokonaan. Tästä huolimatta eksplisiittisiin, implisiittisiin sekä liittyviin voi kaikkiin kuulua virheitä, jotka eivät ole aiottuja virheitä, jolloin ne ovat vahingossa katettu testitapauksilla. Esim. jos testitapauksen kuvaus johtaa virheen löytämiseen, mutta testitapaus ei varsinaisesti odotetuissa tuloksissa käsittele kyseistä virhettä mitenkään, on virhe silloin ns. ”ei-aiottu”.

Ulkopuolisuusluokituksen käyttämisellä voisi olla potentiaalia arvioitaessa testauksen tarvetta ja testauksen onnistumista ylipäätänsä. Se tosin edellyttäisi jatkotutkimusten tekemistä, jotta voitaisiin selvittää kyseisestä metriikasta ainakin: mitä johtopäätöksiä on tehtävissä ohjelmistovirheiden testitapausten avulla löydettyjen virheiden ulkopuolisuuden tasosta? Kenties sen avulla voitaisiin tehdä arvioita testitapausten kattavuudesta, laadusta tai riittävyydestä, esimerkiksi, suuren ulkopuolisten virheiden osuudesta suhteessa testattuihin testitapauksiin. Mikäli löytyneiden virheiden ulkopuolisuudesta voisi tehdä tämän kaltaisia johtopäätöksiä; sitä voitaisiin mahdollisesti hyödyntää arvioitaessa jatkotestauksen tarvetta jo olemassa-olevaan testikattavuuteen tai testitapausten kattavuuden laajentamistarpeen arvioinnissa.

7.3. Ulkopuolisten virheiden läsnäolo

Vaikka virhe olisi testitapausten eksplisiittisellä noudattamisella löydettävissä, tämä koe antoi osviittaa, että näin ei kuitenkaan väistämättä tapahdu. Testitapausten perusteella jotkin virheet olivat selkeästi testitapauksia noudattamalla löydettävissä, mutta siitä huolimatta ne usein jäivät löytymättä. Kun molemmista testatuista ohjelmista löytyi yhteensä 60 eri virhettä, yksi testaaja löysi keskimäärin vain 7,5 virhettä näistä (12,5 % kaikista virheistä), joista oli keskimäärin 3,1 eksplisiittistä, 2,2 implisiittistä, 1,8 liittyvää ja 0,4 täysin ulkopuolista virhettä. Testaajat löysivät siis keskimäärin 51,5 % löydettävissä olevista eksplisiittisistä, 18,1 % implisiittisistä, 6,5 % liittyvistä ja 2,6 % ulkopuolisista virheistä. Kuten voisi olettaa: testitapaukset löytävät enemmän virheitä testitapausten läheisyydestä. Siitä huolimatta vaikka jokin virhe olisi aivan testaajan silmien alla, se siltikin saattaa jäädä tämän huomaamatta. Eksplisiittiset ja implisiittiset virheet ovat käsikirjoituksen kattamia virheitä, jotka pitäisi löytää noudattamalla testitapauksia (varsinkin eksplisiittiset virheet), mutta usein näin ei selvästikään käynyt. *Samojen testitapausten toistamisen (joka on yleinen käytäntö) suoma hyöty saattaa ainakin osittain perustua juuri tähän ilmiöön* (taustaa testitapausten toistamisesta käsitellään luvussa 3.2. Testitapausten merkitys). Ohjelmista löytyneiden virheiden suuri lukumäärä tuli yllätyksenä, vaikkakin uusien virheiden löytymistä osattiin odottaa.

Vertailtaessa ulkopuolisuuden profiilia kahden eri testitapausjoukkojen välillä, on huomioitava, että näillä testitapauksilla oli jo lähtökohtaisesti eri suuret määrät ulkopuoleisia ja sisäpuoleisia virheitä löydettävissä (kuvat 9 ja 10). Tämän vuoksi etäisyyden tuloksia on tulkittava varoen, sillä mahdolliset jakaumat ohjelmien ja testitapausten kombinaatioista eivät ole samat. Raskailla testitapauksilla löydetyt virheet olivat lähempänä testitapauksia ($p < 0{,}001$, $r = 0{,}652$).

Kun ohjelmasta löytyy testitapausten ulkopuolisia virheitä, voi yksi mahdollinen syy tälle yksinkertaisesti olla testitapausten testikattavuuden riittämättömyys testatulle alueelle. Jos se pitää paikkaansa, niin *on mahdollista, että testaustulosten ulkopuolisuuden tasosta voitaisiin tehdä arvioita käytettyjen testitapausten riittävyydestä testatulle ohjelman alueelle, ts., kuinka paljon jatkotestejä tarvitaan?* LibreOfficen kohdalla oli helpompi ajautua testitapausten ulkopuolelle, sillä sen testattava osuus oli selkeästi suurempi ja monimutkaisempi ja siinä ulkopuolisia virheitä oli enemmän kuin Trianglessa.

7.4. Testitapausten rakenteen vaikutus

Testitapausten rakennetta tutkittiin toisella tutkimuskysymyksellä; ”*Vaikuttaako testitapausten testausohjeistuksen rakenne ulkopuolisten virheiden esiintymiseen?”* Kokeen järjestämisen jälkeen havaittiin, että testitapausten rakenteen vertaileminen ei onnistu käytetyllä koeasetelmalla, sillä tässä asetelmassa testitapausten määräämät sisä- ja ulkopuoliset alueet eivät ole samat kevyissä ja raskaissa testitapauksissa (tähän kyllä pyrittiin). Teoriassa vertailun pitäisi kuitenkin olla mahdollista erittäin tarkkaan suunnitellulla koeasetelma, mutta sellaisen asetelman luominen olisi vähintäänkin haasteellista.

Muitakin havaintoja testitapausten rakenteesta kyettiin tekemään testauksesta liittyen kevyisiin ja raskaisiin testitapauksiin kuin vain pelkästään suhteessa ulkopuolisuuden tasoihin. Tosin, tehtäessä tulkintoja testitapausten yksityiskohtaisuudesta, on otettava huomioon, että tässä vertaillaan testitapauksia, joilla oli tietty yksityiskohtaisuuden taso, eli yleistyksen tekeminen; mitä yksityiskohtaisempi sen paremmat/huonommat testauksen tulokset, ei välttämättä ole validi. Liian yksityiskohtaisilla (raskailla) tai liian vähän yksityiskohtaisilla (kevyillä) testitapauksilla saattaa olla oma vaikutuksensa.

Kaikkia kevyitä ja raskaita testitapauksia verrattaessa keskenään, kevyet testitapaukset poikkesivat tilastollisesti merkitsevästi suhteessa raskaisiin testitapauksiin tarkasteltujen muuttujien suhteen lukuun ottamatta väärien virheiden lukumäärää:

- raskailla testitapauksilla löytyi enemmän virheitä ($p < 0,001$, $r = -0,387$),
- raskaiden löytämät virheet olivat vakavampia ($p = 0,002$, $r = -0,295$),
- raskaat testitapaukset menestyivät kevyitä F-arvon suhteen paremmin ($p < 0,001$, $r = -0,290$),
- raskailla testitapauksilla löydetyt virheet olivat lähempänä testitapauksia ($p < 0,001$, $r = 0,652$) ja
- löytyneiden väärien virheiden lukumäärässä ei ollut eroa kevyillä ja raskailla testitapauksilla ($p = 0,294$, $r = -0,099$) (taulukko 7).

Kuitenkin eriteltäessä nämä havainnot ohjelmille erikseen, voidaan havaita Trianglen kohdalla olevan merkitsevämmät p:n arvot kevyiden ja raskaiden testitapausten kohdalla verrattuna LibreOfficen vastaaviin sekä väärillä virheillä puolestaan päinvastoin: p:n arvo oli huomattavasti suurempi. Kevyiden ja raskaiden testitapausten ero siis korostui Trianglessa:

- Trianglen raskaat testitapaukset menestyivät F-arvon suhteen kevyitä paremmin ($p = 0,003$, $r = -0,405$),
- virheitä löytyi raskailla testitapauksilla enemmän Trianglella ($p < 0,001$, $r = -0,478$),
- virheet olivat vakavampia raskailla testitapauksilla Trianglella ($p = 0,001$, $r = -0,431$),
- Trianglesta löytyneiden virheiden etäisyys testitapauksista oli raskailla testitapauksilla pienempi ($p < 0,001$, $r = 0,573$) ja
- väärien virheiden lukumäärässä ei ollut merkitsevää eroa ($p = 0,927$, $r = 0,014$) (taulukko 8).

Samoin kuin Trianglella, myös LibreOfficen raskailla testitapauksilla löytyi enemmän virheitä ($p = 0,045$, $r = -0,269$) sekä virheiden ulkopuolisuuden taso oli kevyillä testitapauksilla raskaita lähempänä testitapauksia ($p < 0,001$, $r = 0,782$) (taulukko 9). F-arvolla ja vakavuudella tulokset eivät kuitenkaan olleet LibreOfficella tilastollisesti merkitseviä toisin kuin Trianglella (F-arvo $p = 0,070$, $r = -0,243$ ja vakavuus $p = 0,242$, $r = -0,157$). Vaikka tilastollista merkittävyyttä ei F-arvolla ja vakavuudella ollut –

tulokset olivat saman suuntaisia. On huomioitava, että ohjelmia tarkastellessa kevyiden ja raskaiden testitapausten suhteen erikseen on otoskoko puolta pienempi, jolloin Wilcoxonin testi ei myöskään ole täysin luotettava tällä pienemmällä aineistolla. Koska tulokset olivat samansuuntaisia sekä kaikkien testitapausten vertailussa oli edelleen havaittavissa tilastollinen merkitsevyys, vaikuttaisi siltä, että suuremmalla otoskoolla voisi saada tilastollisen merkitsevyyden myös LibreOfficelle. Joka tapauksessa, LibreOfficen raskaat testitapaukset eivät menestyneet yhtä hyvin suhteessa kevyisiin kuin Trianglen vastaavat. Mikäli testitapaukset voitaisiin luoda samalla tarkkuudella koko ohjelman laajuudelle kuin mitä Trianglen tapauksessa, raskaat (eli yksityiskohtaisemmat) testitapaukset saattaisivat tuottaa vielä enemmän Trianglen tulosten kaltaiset havainnot. Tätä ei kuitenkaan voida varmuudella todeta, sillä on täysin mahdollista, että kaikkia vaikuttavia muuttujia ei tässä tunneta.

Mikäli molempien ohjelmien testitapaukset edustaisivat yhtäläisesti kevyitä sekä raskaita testitapauksia, niin tällöin tulosten täytyisi vastata enemmän toisiaan. Testitapaukset kattoivat suhteellisesti suuremman osuuden ohjelmasta Trianglen kuin LibreOfficen kohdalla. Yleistyksen tekeminen edellyttäisi merkittävästi laajemman olosuhteiden kirjon, suuremman lukumäärän erilaisia testitapauksia sekä useamman testitapausten kirjoittajan. Siispä näyttäisikin olevan, että pelkästään vertaamalla eri ohjelmien välillä kevyitä ja raskaita testitapauksia, ei voida vielä tehdä riittäviä johtopäätöksiä (ainakaan tässä kokeessa käytetyillä ohjelmilla ja testitapauksilla). Tässä korostuu ongelma yleistyksen tekemisestä verrattaessa testaustuloksia eri ohjelmien välillä. Onkin siis jossain määrin haasteellista puhua ns. yleisestä kevyestä tai raskaasta testitapauksesta, tai ylipäätänsä yleisesti testitapauksesta, sillä tämän kokeen perusteella testitapaukset vaikuttavat olevan kontekstiriippuvaisia eli minkälaista ohjelmaa testataan. Tämä puolestaan herättää kysymyksen ohjelmille tehtyjen tutkimuksen yleistettävyydestä: mikäli tehdään tutkimusta vain muutamalle ohjelmalle – riittääkö se vielä alkuunkaan? Ohjelmia on monenlaisia ja siten on myös oltava erilaisia testitapauksia vastaamaan tätä ohjelmien laajaa kirjoa. Jotta päästäisiin tarkastelemaan tätä kaikkien testitapausten mahdollista joukkoa, tarvittaisiin hyvinkin suuri joukko ohjelmia, jotta saataisiin riittävän laaja-alainen otos erilaisista mahdollisista testitapauksista.

Vaikka tässä tarkastelussa verrataan kahta eri ohjelmaa, kuitenkaan näitä verrattaessa ei verrata pelkästään ohjelmien ominaisuuksia, vaan myös niiden ohjelmien ominaisuuksien ja niille laadittujen testitapausten suhdetta ja sitä kuinka helposti niille ylipäätään on laadittavissa testitapauksia. Esimerkiksi, kompleksisempi ohjelma saattaa vaatia kompleksisempia testitapauksia, jotka puolestaan voivat olla haasteellisempia laatia ja haasteellisemman testitapauksen laatiminen saattaa epäonnistua tai jäädä kokonaan tekemättä.

Mielenkiintoista kyllä, väärien virheiden lukumäärällä ei ollut havaittavaa eroa kevyiden ja raskaiden testitapausten välillä kummallakaan Trianglella ($p = 0{,}9269$, $r = 0{,}014$) eikä LibreOfficella ($p = 0{,}2145$, $r = -0{,}167$). Toisin sanoen, se kuinka yksityiskohtaista tietoa testitapaukset antoivat testattavasta ohjelmasta ja kuinka yksityiskohtaiset testauksen ohjeet annettiin, eivät vaikuttaneet väärien virhehavaintojen löytymiseen. Keskeinen ero kevyiden ja raskaiden testitapausten kohdalla oli odotettujen tulosten puuttuminen kokonaan kevyistä testitapauksista. Koska kevyiden ja raskaiden testitapausten löytämien väärien virheiden lukumäärässä ei ollut merkitsevää eroa, *ainakin tässä kontekstissa oletettujen tulosten puuttuminen testitapauksista ei aiheuta väärien virhehavaintojen tekemistä* (on kuitenkin

huomioitava, että väärille virheille ei tehty vakavuus luokittelua, sillä sitä ei katsottu tarpeelliseksi).

Vaikka raskaat testitapaukset selkeästi menestyivät kevyitä testitapauksia paremmin, niin näiden ero ei kuitenkaan ollut hyvin suuri. Mitä yksityiskohtaisemmat testitapaukset luodaan, sitä työläämpää näiden kirjoittaminen voidaan olettaa olevan, jolloin raskaammista testitapauksista saatava hyöty saadaan lunastettua pidemmän ajan kuluttua, sillä testitapausten luominen on aikaa vievä prosessi [44]. Koska kevyemmät testitapaukset tuottavat myös tuloksia, on niillä jo saatavissa selkeää hyötyä: vähemmälläkin informaatiolla voidaan tuottaa tulosta. Riippuen ohjelmistokehityksen tilasta ja sen aikarajoitteista – molempia lähestymistapoja voitaisiin soveltaa tuloksekkaasti. Esimerkiksi, mikäli halutaan saada nopeita testaustuloksia, voitaisiin testaus aloittaa jo karkeilla testitapauksilla ja siirtyä tästä resurssien salliessa raskaampiin testitapauksiin, jolloin testausta voitaisiin kohdentaa tarkemmin.

7.5. Testauksen dokumentointi

Vaikka nauhoitus ei kuulunut osallistujien itse tekemään virhedokumentaatioon, nauhoitusten käyttäminen virhedokumentoinnissa voisi olla arvokas informaation lähde osana ohjelmistokehitystä. Monien virhemerkintöjen kohdalla virhedokumentaatio oli riittämätön ymmärtämään siinä esitettyä virhettä ja usein pelkkä nauhoitus olisi riittänyt virheen tulkitsemiseksi. Se korosti nauhoituksen tarpeellisuutta osana koetta. Nauhoituksen suoma hyöty virhedokumentaation tukena oli huomattava, mikä omalta osaltaan herättää kysymyksen: ”Voisiko tämän kaltaisia nauhoituksia hyödyntää osana virheiden dokumentointia muuallakin eikä vain ainoastaan osana koejärjestelyä?” Kokeella tavoiteltiin ensisijaisesti löytyneiden virheiden tarkastelua ja erityisesti nauhoituksen avulla huonolaatuisestakin dokumentaatiosta tuli tulkittavissa olevaa. Kerätty nauhoitusmateriaali osoittautui siis hyvinkin tarpeelliseksi. Sanonta; ”Yksi kuva kertoo enemmän kuin tuhat sanaa.”, oli kuvaava tässä yhteydessä. Mikäli samankaltaisia kokeita järjestää, on visuaalisten informaation lähteiden käyttäminen harkitsemisen arvoista. Samasta syystä osana testauksen dokumentaatiota tämän kaltainen visuaalinen informaatio voisi olla mahdollisesti hyödyllinen tuki. Kuitenkin sen sijaan, että nauhoitus olisi koko testaussuorituksen kattava, testaajat voisivat itse tehdä lyhyet nauhoitukset löytämistään virheistä, mikäli se vain olisi mahdollista (kaikista virheistä ei ole mahdollista tehdä nauhoitusta niiden luonteen vuoksi). Tämä olisi helpottanut nauhoitusten läpikäymistä.

Testauksen nauhoitus toi ilmi, että vaikka virhe olisi kirjattu jonkin tietyn testitapauksen kohdalle, se ei suoraan tarkoittanut, että kyseinen virhe olisi löytynyt kyseisellä testitapauksella. Nauhoituksessa joidenkin virheiden dokumentoinnin aikana oli havaittavissa, että testaaja etsi sopivaa testitapausta löytämälleen virheelle ja päätyi lopulta kirjaamaan virheen eri testitapauksen kohdalle, kuin minkä testitapauksen aikana hän löysi virheen. Mahdollisesti osallistujat testasivat testitapauksia osittain lomittain, eikä vain pitäytyneet pelkästään yhden testitapauksen määräämällä alueella.

8. TULEVA TUTKIMUS

Testitapausten testikattavuuden ulkopuolisten virheiden, kuin myös testikattavuuden kattamat virheet, joita ei löydetä, ovat tutkimisen arvoisia ilmiöitä. Tutkimusta voisi lähestyä pyrkimyksellä luoda parempi testitapausten perusrakenne, joka löytäisi paremmin kattavuuden sisäpuoleisia virheitä, tai jopa ulkopuolisia virheitä. Myös itse ulkopuolisuuksien tasojen käyttäminen testitapausten arvioinnissa saattaisi olla potentiaalia sen käyttämisessä testitapausten arvioinnissa. Tätä mahdollisuutta voisi tutkia. Olisi myös mielenkiintoista nähdä, kuinka testausautomaatio suoriutuu eksplisiittisten, implisiittisten, liittyvien ja täysin ulkopuolisten virheiden löytämiseen verrattuna manuaalitestaukseen.

Tehdyssä kokeessa kerättiin informaatiota useasta eri lähteestä, mm. testausnauhoituksista, mikä vaikuttaisi olevan epätavallinen lähestyminen testauksen tutkimukseen. Ulkopuolisten virheiden esiintyminen nostaa esille kysymyksiä testausprosessin kulusta, ja kuinka se vaihtelee yksilöiden välillä: kuinka yksi huomaa eksplisiittisen virheen, mutta toinen ei. Testauksen prosessin on tapahduttava eri tavoin tällöin. Testausta prosessina voisi lähteä tarkastelemaan tarkemmin, ja mallintaa se. Sitä kautta voisi löytyä väylä, jonka kautta testausta pystyttäisiin lähteä parantamaan. Voisiko testausprosessin seurantaan kehittää työkaluja?

8.1. Kokeen toistaminen ja lisäinformaatio testauksesta

Vastaavanlaista koetta soveltaessa, tulisi erityisesti ottaa huomioon työläys, joka liittyy tulosten käsittelyyn. Koetta tehtäessä kävi ilmi, että tulokset on litteroitava, mikäli haluaa tarkkoja tuloksia, sillä testaajat tekivät dokumentoinnissaan paljon virheitä. Mahdollisesti kuitenkin riittävään tarkkuuteen päästäisiin ilmankin, esim. käyttäen koneellista käsittelyä. Huono tasoisen dokumentaation välttämiseksi voisi olla hyvä varmistua testaajien kyvystä tehdä riittävän tasoinen dokumentaatio, esim. harjoituttamalla sitä osallistujille. Pelkkä kokeen alussa tehtävä pikainen opastus ei ole riittävä. Vaihtoehtoisesti olisi mielenkiintoista nähdä kuinka kokeneet testaajat suoriutuisivat testitapauksista.

Parannuksena kokeeseen, olisi myös suotavaa, jotta kaikilla olisi jonkinnäköinen konkreettinen tuntuma testaukseen, jotta osallistujien aikaa ei kuluisi testauksen ymmärtämiseen kokeen aikana. Yksi osallistuja oli jopa kirjoittanut merkintöihin, että alkoi nyt ”vähän vasta ymmärtää”, mitä hänen tulisi tehdä 30 minuutin kohdalle. Osalla tämä saattoi hyvinkin olla ongelma, mikä saattoi selittää osan huonoimmista suorituksista. Ennen kokeen järjestämistä voisikin olla hyödyllistä järjestää harjoittelu testauksesta. Tämän kaltaisella opastamisella saattaisi tosin olla merkittäviä vaikutuksia kokeen lopputuloksiin riippuen mitä testaajille opetettaisiin, mutta jonkin tasoinen kevyt tuntuma testaukseen voisi kuitenkin olla hyödyksi.

Testatun ohjelman virheiden ja testien kattavuuden arvioiminen osoittautui hyvin hankalaksi. Saman kattavuuden tuottaminen raskailla, että kevyillä testitapauksilla näytti varsinkin löytyneiden virheiden valossa lähes mahdottomalta tehtävältä. Pienet erot testitapausten ohjeissa asetti usein löytyneet virheet eri ulkopuolisuuksien tasoille. Siten tasojen määrittäminen olisi tärkeää tehdä suhteessa virheisiin, vaikka tasot olisi

teoriassa määritettävissä pelkästään suhteessa ohjelmaan. Tasojen määrittäminen suoraan ohjelmaan ilman konkreettisia virheitä voi tuottaa epätarkkuutta tasoille.

Nauhoituksen käyttäminen oli erityisen mielenkiintoista ja sitä voitaisiinkin tarkastella lukuisesta eri näkökulmasta. Esimerkiksi, kuinka paljon aikaa käytetään ohjelman testaamiseen verrattuna testitapausten lukuun tai mitä testaaja tekee silloin kun hän löytää ulkopuolisen virheen?

Lisäinformaatiota nauhoitukseen ja testauksen prosessiin liittyen voisi olla hyvä kerätä. Itse nauhoitukseen voisi tehdä näkyväksi kirjainten painallus kuvake, kun valitsee kirjaimen, niin nauhoituksen kulmassa näkyisi, mitä kirjainta testaaja painaa. Lisäksi, hiiren painalluksen havainnollistavat välähdykset tms. Näin nauhoitusta olisi vielä helpompi tulkita. Lisäksi, nauhoituksen tueksi voitaisiin käyttää ääneen ajattelu tutkimusmetodeja [70]. Näin voitaisiin päästä seuraamaan testauksen aikana testaajan ajatuksen juoksua.

Testaajaa voisi olla mielenkiintoista seurata muutoinkin, esim. kehon kieltä, silmien liikettä, pupillien kokoa ja kasvojen ilmeitä testauksen aikana. Tätä kautta saattaisi olla mahdollista luoda uutta kuvaa testauksen prosessista – mitä testaaja oikeasti tekee testatessaan? Jos tähän tietoon päästäisiin käsiksi, saattaisi hyvinkin aueta mahdollisuus kehittää testauksen prosessia. Sen lisäksi, testauksen prosessin seuraaminen testauksen aikana voisi tuottaa lisäinformaatiota testauksesta ja siten siitä ehkä jopa voitaisiin kehittää manuaalisen testauksen työkalu. Ehkäpä konenäköön liittyen voisi tämän kautta kehittää sovelluksia.

8.2. Automaatiotestaus ja manuaalisen testauksen työkalut

Yksimahdollinen tulevaisuuden visio on täysin automatisoitu ohjelmistotestaus. Mikäli tähän halutaan päästä, voi manuaalisen testauksen prosessin takaisinmallinnus tuottaa tulosta, jonka kautta voitaisiin parantaa automaatiotestausta.

Mikäli otettaisiin tilanne, jossa kaikki testitapaukset ajettaisiin testausautomaatiolla, esimerkiksi, on mahdollista, että automaatio löytäisi paremmin eksplisiittiset virheet ohjelmasta, mutta kykeneekö se löytämään implisiittisiä? Toisaalta taas automaatiolla ei välttämättä löytäisi yhtä helposti liittyviä ja ulkopuolisia virheitä. Tätä olisi mahdollista myös tutkia.

Mikäli pääsisi käsiksi informaatiovirtaan, joka syntyy ohjelmistotestauksen yhteydessä, voisi testauksen prosessia päästä tarkastelemaan sen toteutuksen todellisessa ympäristössä. Ongelmana tässä on sen potentiaali kuluttaa testauksen resursseja informaationkeruuseen ja muuhun mittauksiin liittyen. Tähän tarkoitukseen voisi kehittää automatiikan, jonka tausta-ajo ei kuluttaisi resursseja itse testauksesta. Näin olisi myös mahdollista päästä käsiksi helpommin suuriin aineistoihin. Tällä olisi myös esimerkiksi mahdollista mitata aikaa, joka käytetään itse testauksen suorittamiseen ja aikaa, joka kuluu testauksen oheistoimintaan. Testauksen ohella tapahtuva muu toiminta, esimerkiksi: testausprosessien noudattaminen, palaverit, sähköpostien lukeminen ja erinäköiset paperityöt, kuten dokumentaatioiden tekeminen väistämättä vievät resursseja testausaktiviteetilta. Mikä on testausintensiteetin ja testauksen tulosten suhde? Kuinka suureen testausintensiteettiin testauksessa voidaan päästä?

Automaatiota, joka osaisi mitata testauksen suorittamista, esim. testaajan aktiivisuuden avulla, saattaisi olla hyvä mittari testausintensiteetille, josta puolestaan voisi rakentaa mittareita testauksen etenemiselle. Esimerkiksi, automaatio ehkä pystyisi tulkitsemaan, mikäli testaajan keskittyminen on selkeästi herpaantunut jonkin tietyn testitapauksen kohdalla, jolloin kyseinen testitapaus saattaisi olla riittämättömästi testattu. Toisaalta myös testausaktiviteetin seuraamisesta saatettaisiin kyetä voida luoda dokumentaatiota automaattisesti (eli mitä on testattu). Tätä voitaisiin käyttää esim. käyttöliittymän testauksen kattavuuden arviointiin.

8.3. Ulkopuolisuuden tasojen soveltaminen

Ulkopuolisuuden tasojen metriikan (voisi kutsua myös virhe-etäisyydeksi) käyttäminen saattaisi olla hyödyllistä myös muutoinkin testauksessa. Ulkopuolisuusluokituksen käyttämisellä voisi olla potentiaalia arvioitaessa testauksen tarvetta ja testauksen onnistumista ylipäätänsä. Se tosin edellyttäisi jatkotutkimusten tekemistä, jotta voitaisiin selvittää kyseisestä metriikasta ainakin: mitä johtopäätöksiä on tehtävissä ohjelmistovirheiden testitapausten avulla löydettyjen virheiden ulkopuolisuuden tasosta? Kenties sen avulla voitaisiin tehdä arvioita testitapausten kattavuudesta, laadusta tai riittävyydestä, esimerkiksi, suuresta löytyneiden ulkopuolisten virheiden osuudesta suhteessa testattuihin testitapauksiin saatettaisiin voida päätellä testitapaukset puutteellisiksi ohjelman testaamiseksi. Mikäli löytyneiden virheiden ulkopuolisuudesta voisi tehdä tämän kaltaisia johtopäätöksiä; sitä voitaisiin mahdollisesti hyödyntää arvioitaessa jatkotestauksen tarvetta jo olemassa-olevaan testikattavuuteen tai testitapausten kattavuuden laajentamistarpeen arvioinnissa.

Testitapausten sisäpuolisten virheiden löytymättömyydelle on oltava jokin syy, ja yksi mahdollinen selittävä tekijä tälle saattaisi olla testitapauksen riittämätön kyky stimuloida testaajaa. Tätä mahdollisuutta voisi tutkia tarkemmin. Ehkäpä testitapauksia tulisi tarkastella pikemminkin testauksen stimuloijina kuin testien ohjeina. Voisiko tästä löytää jokin vaihtoehtoinen rakenne testitapauksille? Kuinka saada testaaja testaamaan mahdollisimman tuottoisasti? Tätä voisi lähestyä muokkaamalla testitapausten perusrakennetta ja tarkastella sen vaikutusta testaustuloksiin verrattuna tavalliseen testitapauksen rakenteeseen.

9. YHTEENVETO

Tässä opinnäytetyössä tutkittiin testitapausten ulkopuolisten virheiden esiintymistä sekä testausohjeen rakenteen vaikutusta näihin. Tämän tutkimiseksi järjestettiin aikarajoitettu koe manuaalisesta funktionaalisesta käsikirjoitetusta testauksesta. Kokeessa testaajat löysivät virheitä testattujen ohjelmien alueista, jotka olivat testauskäsikirjoituksen ulkopuolella, ts., virheitä, joita ei olisi pitänyt testitapauksilla testattaessa löytyä. Huomattavaa on myös, että keskimäärin henkilöä kohden vain 51,5 % virheistä löydettiin, jotka olisi pitänyt löytyä testitapausten eksplisiittisiä testausohjeita noudattamalla. Ylipäätään virheistä, jotka katsottiin olevan testitapausten kattamalla alueella ohjelmissa; keskimäärin yksi testaaja löysi vain 30,0 % näistä virheistä (51,5 % eksplisiittisistä ja 18,1 % implisiittisistä virheistä löydettiin keskimäärin). Tämän perusteella voidaan sanoa: *vaikka jonkin tietyn testitapauksen testaamisella pitäisi löytää jokin tietty virhe, näin ei kuitenkaan välttämättä käy*. Tästä on erityisesti oltava huolissaan, mikäli luotetaan yksistään testitapausten testikattavuuteen. Ns. riittäväkin testikattavuus ei välttämättä vielä toteudu riittävänä.

Sen lisäksi, että testitapausten kattavan alueen testaus ei välttämättä realisoidu testauksena, osa testauksesta vaikuttaisi kohdentuvan myös testitapausten ulkopuolelle. Ensinnäkin testattujen ohjelmien osuuksien virheiden suuri lukumäärä tuli yllätyksenä, ja ohjelmista löytyikin kaikkiaan 60 uniikkia kelvollista virhettä testaajien toimesta. Yksi testaaja löysi näistä keskimäärin 7,5 virhettä (n. 12,5 % kaikista virheistä) annetun aikarajoituksen puitteissa, joista 71 % oli testitapausten sisäpuolella ja 29 % ulkopuolella. Tämä mukailee Itkosen tekemää havaintoa testitapausten ulkopuolisista virheistä.

Testitapausten testikattavuuden sisä- ja ulkopuolisille virheille on tehtävissä tarkempia määrityksiä, joilla niitä voidaan tarkemmin analysoida. Virheiden analysoimiseksi, virheiden ja testitapausten suhde määriteltiin neljälle ulkopuolisuuden tasolle: 1. eksplisiittinen, 2. implisiittinen, 3. liittyvä ja 4. täysin ulkopuolinen taso. Näistä 1. ja 2. ovat testitapausten testikattavuuden sisäpuolinen osuus ja 3. ja 4. ulkopuolinen osuus. Siten tätä asteikkoa voitaisiin käyttää ordinaalisena virhe-etäisyyden asteikkona. Tämän kaltaiselle luokitukselle saattaisi olla mahdollista kehittää hyödyllisiä sovelluksia, esim. testitapausten soveltuvuuden arvioimiseksi sen tarkoitukseen.

Mitä tulee testitapausten testausohjeiden rakenteen vaikutukselle testaustulosten ulkopuolisuuteen niiden vertaileminen on erittäin haasteellista, sillä ulkopuolisuus on määritettynä suhteessa testitapauksiin, ja kun testitapausten joukot eivät ole samat, ei vertailu myöskään onnistu. Testitapausten testausohjeiden vertailu muilta osin oli mahdollista. Tähän käytettiin Wilcoxonin merkittyjen sijalukujen testiä ja yksityiskohtaisemmat testitapaukset menestyivät selkeästi paremmin karkeisiin testitapauksiin verrattuna.

Rajoituksena kokeelle voidaan katsoa kokeeseen osallistuneiden kokemattomuus, mutta ilman, että testaajat eivät olisi testanneet testitapausten ulkopuolelta; moni virhe olisi jäänyt löytämättä. On myös otettava huomioon, että käytetyllä ohjelmilla ja testaajilla on tunnetusti oma vaikutuksensa kokeiden tuloksiin. Käytettyyn koeasetelmaan liittyen vielä testitapausten muotoilulla, rakenteella ja sanamuodoilla voi olla vaikutuksensa tuloksiin. Näistä rajoitteista huolimatta, testitapausten ulkopuoliselta tuskin voidaan välttyä ja sen parempi huomioiminen olisi aiheellista.

10. LÄHTEET

1. Wiegers K & Beatty J (2013) Software Requirements. Pearson Education.

2. Offutt J (2002) Quality attributes of web software applications. IEEE Software 19(2): 25-32. DOI: https://doi.org/10.1109/52.991329.

3. Kuhn DR, Wallace DR & Gallo AM (2004) Software fault interactions and implications for software testing. IEEE Transactions on Software Engineering 30(6): 418-421. DOI: https://doi.org/10.1109/TSE.2004.24.

4. Ebert C & Dumke R (2007) Software Measurement: Establish-Extract-Evaluate-Execute. Springer Science & Business Media.

5. Grindal M, Offutt J & Mellin J (2006) On the Testing Maturity of Software Producing Organizations. Testing: Academic and Industrial Conference-Practice and Research Techniques, 2006. TAIC PART 2006. Proceedings. IEEE: 171-180. DOI: https://doi.org/10.1109/TAIC-PART.2006.20.

6. Kassab M, DeFranco JF & Laplante PA (2017) Software Testing: The State of the Practice. IEEE Software 34(5): 46-52. DOI: https://doi.org/10.1109/MS.2017.3571582.

7. Tassey G (2002) The economic impacts of inadequate infrastructure for software testing. National Institute of Standards and Technology, RTI Project 7007(011).

8. Myers GJ, Sandler C & Badgett T (2011) The Art of Software Testing. John Wiley & Sons.

9. Jorgensen PC (2016) Software Testing: A Craftsman's Approach. CRC press.

10. Ammann P & Offutt J (2016) Introduction to Software Testing. Cambridge University Press.

11. Itkonen J (2008) Do test cases really matter? An experiment comparing test case based and exploratory testing. Licentiate. Helsinki University of Technology, Department of Computer Science and Engineering.

12. ISO/IEC/IEEE 24765 (2017). Systems and software engineering — Vocabulary. International Organization for Standardization Geneva.

13. Daka E & Fraser G (2014) A Survey on Unit Testing Practices and Problems. Software Reliability Engineering (ISSRE), 2014 IEEE 25th International Symposium on. IEEE: 201-211. DOI: https://doi.org/1f0.1109/ISSRE.2014.11.

14. Bertolino A (2007) Software Testing Research: Achievements, Challenges, Dreams. 2007 Future of Software Engineering. IEEE Computer Society: 85-103. DOI: https://doi.org/10.1109/FOSE.2007.25.

15. Garousi V & Zhi J (2013) A survey of software testing practices in Canada. J Syst Software 86(5): 1354-1376. DOI: https://doi.org/10.1016/j.jss.2012.12.051.

16. Paasivaara M, Lassenius C & Heikkilä VT (2012) Inter-Team Coordination in Large-Scale Globally Distributed Scrum: Do Scrum-of-Scrums really Work? Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement. ACM: 235-238. DOI: https://doi.org/10.1145/2372251.2372294.

17. Hellmann TD, Sharma A, Ferreira J & Maurer F (2012) Agile Testing: Past, Present, and Future--Charting a Systematic Map of Testing in Agile Software Development. 2012 Agile Conference. IEEE: 55-63. DOI: https://doi.org/10.1109/Agile.2012.8.

18. Nerur S, Mahapatra R & Mangalaraj G (2005) Challenges of migrating to agile methodologies. Communications of the ACM - Adaptive complex enterprises. ACM 48(5): 72-78. DOI: http://dx.doi.org/10.1145/1060710.1060712.

19. Stoica M, Mircea M & Ghilic-Micu B (2013) Software Development: Agile vs. Traditional. Informatica Economica 17(4). DOI: https://doi.org/10.12948/issn14531305/17.4.2013.06.

20. Haberl P, Spillner A, Vosseberg K & Winter M (2011) Software Test in Practice. International Software Testing Qualifications Board: German Testing Board. URL: https://www.istqb.org/documents/Survey_GTB.pdf. Accessed 10.11.2019

21. Garousi V & Varma T (2010) A replicated survey of software testing practices in the Canadian province of Alberta: What has changed from 2004 to 2009? Journal of Systems and Software 83(11): 2251-2262. DOI: https://doi.org/10.1016/j.jss.2010.07.012.

22. Ng SP, Murnane T, Reed K, Grant D & Chen TY (2004) A Preliminary Survey on Software Testing Practices in Australia. Australian Software Engineering Conference, 2004. Proceedings. IEEE: 116-125. DOI: https://doi.org/10.1109/ASWEC.2004.1290464.

23. Causevic A, Sundmark D & Punnekkat S (2010) An Industrial Survey on Contemporary Aspects of Software Testing. Software Testing, Verification and Validation (ICST), 2010 Third International Conference on. IEEE: 393-401. DOI: https://doi.org/10.1109/ICST.2010.52.

24. Rafi DM, Moses KRK, Petersen K & Mäntylä M (2012) Benefits and Limitations of Automated Software Testing: Systematic Literature Review and Practitioner Survey. Proceedings of the 7th International Workshop on Automation of Software Test. IEEE Press: 36-42. DOI: https://doi.org/10.1109/IWAST.2012.6228988.

25. Lee J, Kang S & Lee D (2012) Survey on software testing practices. IET software 6(3): 275-282. DOI: https://doi.org/10.1049/iet-sen.2011.0066.

26. Kasurinen J, Taipale O & Smolander K (2010) Software test automation in practice: empirical observations. Advances in Software Engineering 2010. DOI: http://dx.doi.org/10.1155/2010/620836.

27. ISO/IEC/IEEE 29119-1 (2013). Software and systems engineering-Software testing-Part 1: Concepts and definitions.

28. Henard C, Papadakis M, Harman M, Jia Y & Le Traon Y (2016) Comparing White-Box and Black-Box Test Prioritization. 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE). IEEE: 523-534. DOI: https://doi.org/10.1145/2884781.2884791.

29. Garousi V & Mäntylä MV (2016) A systematic literature review of literature reviews in software testing. Information and Software Technology 80: 195-216. DOI: https://doi.org/10.1016/j.infsof.2016.09.002.

30. Hamlet R (1989) Theoretical Comparison of Testing Methods. ACM SIGSOFT Software Engineering Notes. ACM 14: 28-37. DOI: https://doi.org/10.1145/75308.75313.

31. Garousi V, Felderer M & Kılıçaslan FN (2018) A survey on software testability. Information and Software Technology. DOI: https://doi.org/10.1016/j.infsof.2018.12.003.

32. Orso A & Rothermel G (2014) Software Testing: A Research Travelogue (2000–2014). Proceedings of the on Future of Software Engineering. ACM: 117-132. DOI: https://doi.org/10.1145/2593882.2593885.

33. World Quality Report 2018/19 (2019). Capgemini, Sogeti & Micro Focus. URL: https://www.sogeti.com/solutions/testing/wqr/ Accessed 9.10.2019.

34. International Software Testing Qualifications Board. URL: https://www.istqb.org/ Accessed 6.6.2018.

35. American Society for Quality (ASQ) - Certification Catalog. URL: http://videos.asq.org/asq-certification-at-50 Accessed 6.6.2018.

36. Quality Assurance International (QAI) - Certifications. URL: https://www.qaiglobalinstitute.com/about-us/ Accessed 6.6.2018.

37. International Software Testing Qualifications Board - Foundation Level Content. URL: https://www.istqb.org/certification-path-root/foundation-level/foundation-level-content.html Accessed 3.4.2018.

38. International Software Testing Qualifications Board - Foundation Level Exam Structure. URL: https://www.istqb.org/certification-path-root/foundation-level/foundation-level-exam-structure.html Accessed 3.4.2018.

39. Finnish Software Testing Board - ISTQB sertifiointi. URL: http://www.fistb.fi/fi/sertifiointi Accessed 3.4.2018.

40. Page A, Johnston K & Rollison B (2008) How we Test Software at Microsoft. Microsoft Press.

41. Mäntylä MV & Itkonen J (2013) More testers–The effect of crowd size and time restriction in software testing. Information and Software Technology 55(6): 986-1003.

42. The Invisible Gorilla. URL: http://www.theinvisiblegorilla.com/ Accessed 1.6.2018.

43. Mack A & Rock I (1998) Inattentional Blindness. MIT press.

44. Niittyviita S (2016) The Cost Efficiency of Exploratory Testing: ISTQB Certified Testing Compared with RST. Bachelor's thesis. University of Oulu, Department of Computer Science and Engineering.

45. Banerjee I, Nguyen B, Garousi V & Memon A (2013) Graphical user interface (GUI) testing: Systematic mapping and repository. Information and Software Technology 55(10): 1679-1694. DOI: https://doi.org/10.1016/j.infsof.2013.03.004.

46. Groce A, Alipour MA & Gopinath R (2014) Coverage and its Discontents. Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software. ACM: 255-268. DOI: https://doi.org/10.1145/2661136.2661157.

47. Gligoric M, Groce A, Zhang C, Sharma R, Alipour MA & Marinov D (2015) Guidelines for coverage-based comparisons of non-adequate test suites. ACM Transactions on Software Engineering and Methodology (TOSEM) 24(4): 22. DOI: https://doi.org/10.1145/2660767.

48. Gligoric M, Groce A, Zhang C, Sharma R, Alipour MA & Marinov D (2013) Comparing Non-Adequate Test Suites using Coverage Criteria. Proceedings of the 2013 International Symposium on Software Testing and Analysis. ACM: 302-313. DOI: https://doi.org/10.1145/2483760.2483769.

49. Gopinath R, Jensen C & Groce A (2014) Code Coverage for Suite Evaluation by Developers. Proceedings of the 36th International Conference on Software Engineering. ACM: 72-82. DOI: https://doi.org/10.1145/2568225.2568278.

50. Ostrand TJ & Weyuker EJ (2002) The distribution of faults in a large industrial software system. ACM SIGSOFT Software Engineering Notes 27(4): 55-64. DOI: https://doi.org/10.1145/566171.566181.

51. Radjenović D, Heričko M, Torkar R & Živkovič A (2013) Software fault prediction metrics: A systematic literature review. Information and software technology 55(8): 1397-1418. DOI: https://doi.org/10.1016/j.infsof.2013.02.009.

52. Memon AM, Soffa ML & Pollack ME (2001) Coverage criteria for GUI testing. ACM SIGSOFT Software Engineering Notes 26(5): 256-267. DOI: https://doi.org/10.1145/503271.503244.

53. Briand L & Pfahl D (1999) Using Simulation for Assessing the Real Impact of Test Coverage on Defect Coverage. Proceedings 10th International Symposium on

Software Reliability Engineering. IEEE: 148-157. DOI: https://doi.org/10.1109/ISSRE.1999.809319.

54. Staats M, Gay G, Whalen M & Heimdahl M (2012) On the Danger of Coverage Directed Test Case Generation. International Conference on Fundamental Approaches to Software Engineering. Springer: 409-424. DOI: https://doi.org/10.1007/978-3-642-28872-2_28.

55. Gay G, Staats M, Whalen M & Heimdahl M (2015) The risks of coverage-directed test case generation. IEEE Transactions on Software Engineering 41(8): 803-819. DOI: https://doi.org/10.1109/TSE.2015.2421011.

56. Inozemtseva L & Holmes R (2014) Coverage is Not Strongly Correlated with Test Suite Effectiveness. Proceedings of the 36th International Conference on Software Engineering. ACM: 435-445. DOI: https://doi.org/10.1145/2568225.2568271.

57. Prakash V & Gopalakrishnan S (2011) Testing Efficiency Exploited: Scripted Versus Exploratory Testing. 2011 3rd International Conference on Electronics Computer Technology. IEEE 3: 168-172. DOI: https://doi.org/10.1109/ICECTECH.2011.5941824.

58. Itkonen J, Mantyla MV & Lassenius C (2007) Defect Detection Efficiency: Test Case Based Vs. Exploratory Testing. First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007). IEEE: 61-70. DOI: https://doi.org/10.1109/ESEM.2007.56.

59. Afzal W, Ghazi AN, Itkonen J, Torkar R, Andrews A & Bhatti K (2015) An experiment on the effectiveness and efficiency of exploratory testing. Empirical Software Engineering 20(3): 844-878. DOI: https://doi.org/10.1007/s10664-014-9301-4.

60. do Nascimento LH & Machado PD (2007) An Experimental Evaluation of Approaches to Feature Testing in the Mobile Phone Applications Domain. Workshop on Domain specific approaches to software test automation: in conjunction with the 6th ESEC/FSE joint meeting. ACM: 27-33. DOI: https://doi.org/10.1145/1294921.1294926.

61. LibreOffice. URL: https://www.libreoffice.org/ Accessed 30.3.2018.

62. Greenshot. URL: https://getgreenshot.org/ Accessed 10.4.2018.

63. Open Broadcaster Software. URL: https://obsproject.com/ Accessed 12.4.2018.

64. IEEE 1044-2009 (2010). Standard Classification for Software Anomalies.

65. Manning C, Raghavan P & Schütze H (2010) Introduction to information retrieval. Cambridge University Press.

66. Sasaki Y (2007) The truth of the F-measure. Teach Tutor mater 1(5): 1-5.

67. Chen TY, Kuo F & Merkel R (2004) On the Statistical Properties of the F-Measure. Fourth International Conference on Quality Software, 2004. QSIC 2004. Proceedings. IEEE: 146-153. DOI: https://doi.org/10.1109/QSIC.2004.1357955.

68. Itkonen J & Mäntylä MV (2014) Are test cases needed? Replicated comparison between exploratory and test-case-based software testing. Empirical Software Engineering 19(2): 303-342. DOI: https://doi.org/10.1007/s10664-013-9266-8.

69. Camilo F, Meneely A & Nagappan M (2015) Do Bugs Foreshadow Vulnerabilities? A Study of the Chromium Project. Proceedings of the 12th Working Conference on Mining Software Repositories. IEEE Press: 269-279. DOI: https://doi.org/10.1109/MSR.2015.32.

70. Jääskeläinen R (2010) Think-aloud protocol. Handbook of translation studies 1: 371-374. John Benjamins Publishing Company.

11. LIITTEET

Liite 1 Testitapaukset – ryhmä A
Liite 2 Testitapaukset – ryhmä B
Liite 3 Ohjelmistovirheiden dokumentointi pohja
Liite 4 Ohjelmista löytyneet virheet ja niiden luokitukset

Triangle test cases

Test Case#	Test Title	Test Summary	Test Steps	Expected Result
1	Instruction text	Check that instruction text is sufficient to use the software	-	The instructions are comprehensible
2	Text input: "Side A"	Verify that text box can be selected and values can be entered to the text box	1. Select the text box 2. Enter values to it 3. Edit the values	- Text box can be selected - Values can be entered and edited
3	Text input: "Side B"	Verify that text box can be selected and values can be entered to the text box	1. Select the text box 2. Enter values to it 3. Edit the values	- Text box can be selected - Values can be entered and edited
4	Text input: "Side C"	Verify that text box can be selected and values can be entered to the text box	1. Select the text box 2. Enter values to it 3. Edit the values	- Text box can be selected - Values can be entered and edited
5	CHECK -button functionality	Verify that CHECK -button can be pressed	1. Select CHECK, while Side A, B and C are empty 2. Select CHECK, while Side A, B and C have values	- Results text is drawn - Bottom triangle is drawn if valid values are entered to Side A, Side B and Side C
6	Results I: Equilateral	Verify that equilateral text is displayed (all sides of the triangle are equal)	1. Enter Side A, Side B, Side C, with three equal values 2. Select CHECK	Equilateral text is displayed in "Results" text field.
7	Results II: Isosceles	Verify that isosceles text is displayed (two of the sides are equal)	1. Enter Side A, Side B, Side C, with two of the values being equal 2. Select CHECK	Isosceles text is displayed in "Results" text field.
8	Results III: Scalene	Verify that scalene text is displayed (no side is equal to any of the others)	1. Enter Side A, Side B, Side C, with none of the values being equal 2. Select CHECK	Scalene text is displayed in "Results" text field.
9	Draw triangle I: Equilateral	Verify that equilateral triangles are drawn correctly	1. Enter Side A, Side B, Side C, with three equal values 2. Select CHECK	Equilateral triangle is drawn and scaled down to fit to inside the rectangle at the bottom of the window.
10	Draw triangle II: Isosceles	Verify that isosceles triangles are drawn correctly	1. Enter Side A, Side B, Side C, with two of the values being equal 2. Select CHECK	Isosceles triangle is drawn and scaled down to fit inside the rectangle at the bottom of the window.
11	Draw triangle III: Scalene	Verify that scalene triangle is drawn correctly	1. Enter Side A, Side B, Side C, with none of the values being equal 2. Select CHECK	Scalene triangle is drawn and scaled down to fit inside the rectangle at the bottom of the window.
12	Draw triangle IV: No triangle	Verify that no triangle will be drawn if triangle is not scalene, isosceles or equilateral	1. Enter such values to A, B and C that no triangle can be drawn from them 2. Select CHECK	Bottom rectangle should be empty
13	Value restrictions	Verify that only positive values are accepted for triangle sides	-	Only positive values should be accepted as triangle side values.
14	Value range	Test that decimal values and integers are accepted as values	-	Both integers and decimal values are accepted as values.

15	Non-numeric values	Test that only numeric values are accepted	-	Triangle cannot be drawn using non-numeric values.
16	Window scaling	Test that window is scaled correctly	-	Window can be scaled.

LibreOffice test cases

Test Case#	Test Title	Test Summary	Test Steps	Expected Result
17	Access 'Special Characters' I	Verify access using 'Insert' -dropdown	1. Select 'Insert' from toolbar (alternatively using the hotkey alt+I) 2. Select 'Special characters' from the dropdown	-
18	Access 'Special Characters' II	Verify access using 'Insert' -dropdown	1. Select 'Insert' from toolbar (alternatively using the hotkey alt+I) 2. Select 'Special characters' using hotkey 'p'	-
19	Access 'Special Characters' III	Verify access using toolbar	1. Select "Ω-button" from libre office toolbar 2. Select 'More characters'	-
20	Access 'Special Characters' IV	Verify access using toolbar	1. Select "Ω-button" from libre office toolbar 2. Select the hotkey 'M'	-
21	Special characters toolbar	Test 'Special characters' functionality from toolbar (Ω-button).	-	-
22	Special character search field	Test search field	-	-
23	Hotkeys in special characters	Test all hotkeys for special character page	-	-
24	Font	Test the font selection	-	-
25	Subset	Test the subset selection	-	-
26	Special character catalogue	Test special character catalogue (grid). The special characters are in UNICODE format. https://unicode-table.com/en/	-	-
27	Recent characters	Test recent characters	-	-

28	Favourite characters	Test favourite characters	-	-
29	Help	Test help functionality	-	-
30	Special character image	Test special character image	-	-
31	Hexadecimal	Test hexadecimal text box	-	-
32	Decimal	Test decimal text box	-	-
33	Opening and closing	Test opening special characters dialog and closing it	-	-
34	Right click menu	Test the functionalities of the right click menu.	-	-
35	Insert	Test inserting to document	-	-

Triangle test cases

Test Case#	Test Title	Test Summary	Test Steps	Expected Result
1	Instruction text	Check instruction text for defects	-	-
2	Text inputs: "Side A", "Side B" and "Side C"	Test the functionality of text boxes	-	-
3	CHECK - button functionality	Test the CHECK button	-	-
4	Value testing	1. Test equilateral, isosceles and scalene triangles using different values. Triangle definitions: Equilateral=equal sides, Isosceles=Two equal sides, Scalene=no equal sides 2. Test that only positive values are accepted.	-	-
5	Results display	Test that results displays correct information	-	-
6	Draw box	Verify that scalene text is displayed (no side is equal to any of the others)	-	-
7	Triangle 2000 window	Test Triangle 2000 window	-	-

LibreOffice test cases

Test Case#	Test Title	Test Summary	Test Steps	Expected Result
8	Access 'Special Characters' I	Verify access using 'Insert' -dropdown	1. Select 'Insert' from toolbar (alternatively using the hotkey alt+I) 2. Select 'Special characters' from the dropdown	Special Characters feature is accessed.
9	Access 'Special Characters' II	Verify access using 'Insert' -dropdown	1. Select 'Insert' from toolbar (alternatively using the hotkey alt+I) 2. Select 'Special characters' using hotkey 'p'	Special Characters feature is accessed.
10	Access 'Special Characters' III	Verify access using toolbar	1. Select "Ω-button" from libre office toolbar 2. Select 'More characters'	Special Characters feature is accessed.
11	Access 'Special Characters' IV	Verify access using toolbar	1. Select "Ω-button" from libre office toolbar 2. Select the hotkey 'M'	Special Characters feature is accessed.

12	Search field I	Verify filtering functionality of the search field.	Search using different inputs	- The search results show the unicode characters matching the search inputs - Only the characters are shown, which match the search criteria.
13	Font I	Verify that fonts can be selected from 'font' drop-down	Select different fonts from 'font' -dropbox	Fonts can be selected from the dropbox
14	Font scroll I	Test the font's drop-down functionalities	1. Use the dropbox arrows to move between the items in the scrollbar 2. Scroll using the middle mouse button's scroll wheel	- Dropbox's arrows can be used to navigate in the dropbox - Dropbox can be moved using the scroll wheel
15	Font scroll II	Test the scrolling functionality of 'font selection dropbox'	1. Move mouse cursor over "Font" -dropbox without selecting it 2. Scroll using middle mouse buttons scroll wheel	The shown "font group" is changed when scrolling without selecting the dropbox.
16	Subset scroll I	Test the scrolling functionality of 'subset selection dropbox'	1. Move mouse cursor over "Subset" -dropbox without selecting it 2. Scroll using middle mouse buttons scroll wheel	The shown "subset group" is changed when scrolling.
17	Font scroll + Search I	Test that scrolling between 'fonts' can be combined with the filter	1. Enter a search criteria 2. Move mouse over the 'font' -dropbox 3. Scroll using mouse wheel	When scrolling between different fonts, the search filter is applied
18	Subset scroll + Search I	Test scrolling between 'subsets' can be combined with the filter	1. Enter a search criteria 2. Move mouse over the 'subset' -dropbox 3. Scroll using mouse wheel	When scrolling between different fonts, the search filter is applied
19	Special character selection	Verify the selection of special characters	1. Select one special character 2. Select another special character	A character is highlighted when selected.
20	Special character unicode as text	Verifying that special character unicodes are shown in the right side of the 'Special Characters' window.	1. Select an unicode character from the list 2. Compare the selected character's displayed unicode with the unicode characters https://unicode-table.com/en/#0001	- Unicode character name is displayed, - The unicode name matches with the unicode table.
21	Special character unicode value	Check that special character unicode values are shown below special character value in the context menu.	1. Select an unicode character from the list 2. Compare the selected character's displayed unicode value with the unicode character's value https://unicode-table.com/en/#0001	- The displayed unicode's numeric value matches with the unicode table. - Unicode value is shown under the unicode character's name.

22	Special character image	Verify that the selected special characters are shown in a box, right side of Special Characters window.	1. Select an unicode character from the list	The selection matches the shown character.
23	Unicode name	Test that the correct special unicode character is shown	Select an unicode character from the list	The unicode name of the selected character is shown below the character image
24	Add to favorites I	Test adding characters to favorites using 'add to favorites'	1. Select character 2. Select 'Add to Favorites' -button	- After the character is added to favorites, the button changes to "Remove from Favorites" - Characters can be added to favorites using 'Add to Favorites'
25	Add to favorites II	Test adding characters to favorites using right mouse button	1. Select character using right mouse button 2. Select 'Add to Favorites' from the drop down and the keyboard hotkey "A"	- After the character is added to favorites, the button changes to "Remove from Favorites" - Characters can be added to favorites using the hotkey "A" - Characters can be added to favorites from the drop-down
26	Add to favorites III	Verify the limitations of the number of characters in 'Favorite characters'	1. Remove all favorite characters 2. Add characters one by one up until there are 14 characters in 'favorite characters'	Favorite characters can have 0-14 characters in 'favorite characters'.
27	Add to favorites IV	Test add to favorites when favorites has maximum number of characters	1. Add characters to 'favorite characters' until it is full (14 characters in total) 2. Try to add more characters to favorites	14 characters as favorites is the maximum.
28	Remove from favorites I	Test removal from favorites using 'Remove from Favorites' button	1. Select character from 'Favorite Characters' list 2. Remove them by using 'Remove from favorites' button	Character(s) is removed from favorites.
29	Remove from Favorites II	Test removal from favorites using RMB	1. Right click a character from 'Favorite characters' list 2. Select 'Remove' to remove 3. Repeat using the hotkey 'R' to remove	Character(s) is removed from favorites.
30	Remove from Favorites III	Test removal from favorites using RMB from "main character list"	1. Right click a character from "main character list" 2. Select 'Remove' to remove 3. Repeat using the hotkey 'R' to remove	Character(s) is removed from favorites.
31	Help access	Verify access "Help" using 'Help' -button	-	Help is accessed.

32	Insert character I	Test insert using right mouse button (RMB)	1. Select a character using right mouse button 2. Select 'Insert into document' 3. Repeat using the hotkey for insert 'I'	- The selected character(s) is inserted into the document. - The character is added to "recent characters"
33	Insert character II	Test insert using right mouse button+hotkey 'I'	1. Select a character using right mouse button 2. Select 'I'	- The selected character(s) is inserted into the document. - The character is added to "recent characters"
34	Insert character III	Test insert using 'Insert' button	1. Select a character using left mouse button 2. Select 'Insert' button	- The selected character(s) is inserted into the document. - The character is added to "recent characters"
35	Insert character IV	Test insert from "Recent Characters"	1. Select "Ω-button" from libre office toolbar 2. Select character(s) from 'Recent characters'	The selected character(s) is inserted into the document.
36	Insert character V	Test insert from "Favorite Characters"	1. Select "Ω-button" from libre office toolbar 2. Select character(s) from 'Favorite characters'	The selected character(s) is inserted into the document.
37	Hexadecimal I	Test hexadecimal display when selecting	1. Select special characters	-Hexadecimal value is shown correctly
38	Hexadecimal II	Test search using hexadecimal text box	1. Enter values to the text box	-Hexadecimal value text box can be used for searching special characters -Hexadecimal value displayed matches the decimal value -Correct special character is selected using The search
39	Decimal I	Test decimal text box	1. Select special characters	-Decimal value is shown correctly
40	Decimal II	Test search using decimal text box	1. Enter values to the text box	-Decimal value text box can be used for searching special characters -Eecimal value displayed matches the hexadecimal value -Correct special character is selected using The search
41	Copy clipboard	Test copy to clipboard. 'Copy clipboard' can be opened by right clicking a special character.	-	Copied characters can be pasted.

42	Character listing I	Verify the UNICODE characters grid.	-	Characters are displayed in the correct order.
43	Character listing II	Verify that the scrollbar for characters can be used.	-	Scrollbar can be used to change the characters, which are seen.
44	Character listing III	Test that arrow keys can be used to change the selected character.	-	The selected character(s) is changed.

1. **Virheen otsikko**

Ohjelmistovirheen kuvaus
[Kirjoita tähän mikä virhe on ja miksi se on virhe.]

Testitapauksen numero
[Testitapauksen nro.]

Virheen havaitsemisen aika
[00:00]

Kuvaus	Suhteellinen vakavuus (suurempi luku vakavampi)	Ulkopuolisuus A-ryhmälle	Ulkopuolisuus B-ryhmälle	Ohjelma	Virheen löytäneiden opiskelijoiden lukumäärä
Kolmion mahtuminen alueelleen	3	Eksplisiittinen	Implisiittinen	Triangle	34
Negatiiviset arvot hyväksytään syötekenttään, vaikka sen ei pitäisi olla mahdollista	4	Eksplisiittinen	Eksplisiittinen	Triangle	39
Ei-numeerisia arvoja ei tulisi hyväksyä	3	Eksplisiittinen	Implisiittinen	Triangle	23
Ikkunan skaalautuminen	1	Eksplisiittinen	Eksplisiittinen	Triangle	24
Suuret luvut tuottavat ongelmia laskentaan	4	Liittyvä	Liittyvä	Triangle	9
Viiva piirtyy ulkopuolelle: Esim. 1+2+3	4	Implisiittinen	Implisiittinen	Triangle	20
Fatal error arvolla 6766	4	Täysin ulkopuolinen	Täysin ulkopuolinen	Triangle	1
Kolmio piirretään, vaikka sitä ei pitäisi	4	Implisiittinen	Implisiittinen	Triangle	13
Check-nappi voi jäädä jumiin	3	Liittyvä	Liittyvä	Triangle	1
Pilkku ei toimi desimaali erottimena	3	Implisiittinen	Täysin ulkopuolinen	Triangle	15
Pilkun ja pisteen käytöstä tulisi mainita	1	Liittyvä	Liittyvä	Triangle	1
Ohje teksti on liian epäselvä	2	Eksplisiittinen	Implisiittinen	Triangle	1
Piirtää vääränlaisen kolmion: sivut piirtyvät väärin	4	Implisiittinen	Implisiittinen	Triangle	6
Maalattu teksti ja ctrl+c, ctrl+v pikanäppäinten kanssa ei toimi oikein	1	Täysin ulkopuolinen	Täysin ulkopuolinen	Triangle	3
Kolmion piirto ilman yhtä arvoa aiheuttaa virheen	4	Liittyvä	Liittyvä	Triangle	1
Tab+nuolinäppäin yhdistelmä liikutti koko ikkunaa	2	Täysin ulkopuolinen	Täysin ulkopuolinen	Triangle	1
Voi käyttää useampaa desimaalia	2	Liittyvä	Täysin ulkopuolinen	Triangle	1

Pikanäppäimiä ei voi käyttää search-valikon ollessa auki	3	Implisiittinen	Implisiittinen	LibreOffice	11
Subset pikanäppäin ei toimi. "S" on Search-pikanäppäin subset-pikanäppäimen sijasta.	2	Liittyvä	Täysin ulkopuolinen	LibreOffice	2
Fontilla ei ole pikanäppäintä	1	Liittyvä	Täysin ulkopuolinen	LibreOffice	1
Heksadesimaali kentän käyttäminen: esikatselu ei päivity ja dokumenttiin liitetään valittuna ollut erikoismerkki	4	Liittyvä	Eksplisiittinen	LibreOffice	31
Desimaali kentän käyttäminen: esikatselu ei päivity ja dokumenttiin liitetään valittuna ollut erikoismerkki	4	Liittyvä	Eksplisiittinen	LibreOffice	27
Erikoismerkkien haku omilla merkeillään ei toimi	2	Liittyvä	Implisiittinen	LibreOffice	25
Hyppivä kursori: hexadecimal- ja decimal-laatikko ja ei-numero merkkien syöttö.	1	Liittyvä	Liittyvä	LibreOffice	4
Subset-valikko ei toimi, kun hakuun on syötetty tekstiä	3	Täysin ulkopuolinen	Eksplisiittinen	LibreOffice	19
Haku ei poista valittavissa olevia fontteja	3	Täysin ulkopuolinen	Implisiittinen	LibreOffice	4
Heksadesimaalien haku ei toimi, mikäli normaali haussa on tekstiä	2	Täysin ulkopuolinen	Liittyvä	LibreOffice	1
Desimaali hakukentän tyhjennys asettaa default-luvuksi 13 siihen	2	Liittyvä	Liittyvä	LibreOffice	1
Sisäkkäiset erikoismerkki ikkunat. (Special characters)	1	Liittyvä	Täysin ulkopuolinen	LibreOffice	5
Oikean painalluksen kontekstivalikko ei toimi työkalupalkissa.	4	Liittyvä	Liittyvä	LibreOffice	13
Toolbarin oikea valikko ei sulkeudu painamalla ohi siitä.	1	Liittyvä	Liittyvä	LibreOffice	4
Toisen valikon avaus, kun toolbar valikko on	2	Täysin ulkopuolinen	Täysin ulkopuolinen	LibreOffice	1

avattuna aiheuttaa focus ongelman					
Toolbar korostus ei toimi.	2	Liittyvä	Liittyvä	LibreOffice	1
Toolbar-valikossa ei voi liikkua nuolinäppäimillä. Ensimmäinen kirjain näkyy valittuna. Enter syöttää seuraavan kirjaimen siitä.	1	Liittyvä	Liittyvä	LibreOffice	3
Oikealla hiiren napilla poistettuaan toolbarista special charactersin toolbar lakkasi toimimasta oikean hiiren napin toiminnolla kokonaan	4	Liittyvä	Liittyvä	LibreOffice	1
Jotkin merkit menevät päällekkäin	3	Liittyvä	Liittyvä	LibreOffice	1
Arabialaisten kirjainten syöttö ei toimi aina oikein: välillä teksti syötetään väärälle puolelle latinalaisten kirjainten kanssa.	3	Liittyvä	Liittyvä	LibreOffice	1
Oikealle painaminen näppäimistöstä liikuttaa kursoria vasemmalle arabialaisilla kirjaimilla	2	Täysin ulkopuolinen	Täysin ulkopuolinen	LibreOffice	1
Insert painaminen sulkee special characters ikkunan	2	Implisiittinen	Implisiittinen	LibreOffice	3
Fontti: Microsoft office preview on hankala ja sotkuinen	2	Liittyvä	Liittyvä	LibreOffice	4
Puuttuva merkki	2	Liittyvä	Liittyvä	LibreOffice	5
Merkki ei vastaa oikeaa unicode merkkiä	3	Liittyvä	Liittyvä	LibreOffice	2
Fontin skrollaus ei vaihda päänäkymän fontteja skrollauksen aikana. Vaatii erillisen napautuksen fontista, jotta se vaihtuisi.	1	Liittyvä	Implisiittinen	LibreOffice	5
Subset skrollaus liikuttaa samalla kahta ensimmäistä merkkiä merkki ikkunasta	1	Liittyvä	Implisiittinen	LibreOffice	1
Muutettu font ei säily: Font vaihtuu takaisin valitsemalla se uudestaan (Ulkopuoleinen font)	4	Täysin ulkopuolinen	Täysin ulkopuolinen	LibreOffice	2

Toinen samaa merkkiä eri fontilla korvaa toisen favorites listasta	3	Liittyvä	Liittyvä	LibreOffice	1
Help linkki ei ohjaa suoraan help sivulle	1	Implisiittinen	Liittyvä	LibreOffice	3
"What is this" ei toimi ollenkaan.	4	Täysin ulkopuolinen	Täysin ulkopuolinen	LibreOffice	1
Pdf ja docx insert ei toimi	4	Täysin ulkopuolinen	Täysin ulkopuolinen	LibreOffice	2
Maksimimäärä merkkejä ei täsmää testitapauksen kanssa. (14 ja 16)	1	Täysin ulkopuolinen	Eksplisiittinen	LibreOffice	26
Add to favorites-teksti voi näkyä, vaikka kirjain olisi jo favorites listassa	2	Liittyvä	Liittyvä	LibreOffice	1
Kun avaa näkymän ja valitsee 'add to favorites', niin merkkiä ei lisätä suosikkeihin	2	Liittyvä	Liittyvä	LibreOffice	1
Kuva ei päivity vaihdettaessa subset ryhmää	1	Implisiittinen	Implisiittinen	LibreOffice	1
Skaalautuminen ei ole sulavaa ja toimii merkittävän hitaasti	3	Täysin ulkopuolinen	Täysin ulkopuolinen	LibreOffice	1
Ikkunan koon muuttaminen ei toimi hiiren oikealla näppäimellä	3	Täysin ulkopuolinen	Täysin ulkopuolinen	LibreOffice	1
Fontti hukkuu käytettäessä leikepöytää. (Copy to clipboard)	4	Liittyvä	Liittyvä	LibreOffice	1
Ei voi avata kahta merkkiä peräjälkeen edellisen valikon ollessa vielä auki oikealla hiiren painikkeella	1	Liittyvä	Liittyvä	LibreOffice	1
Välkkyvä UI tehdessä asioita	3	Implisiittinen	Implisiittinen	LibreOffice	6
Väärä kuva valinnan aikana. (pieni kuva ei vastaa oikeaa kuvaa, kun sen valitsee)	1	Liittyvä	Eksplisiittinen	LibreOffice	1
Valinta, search, font ja hexa decimaali viittaavat kaikki eri lukuihin. Sisältää myös virheellistä informaatiota.	3	Liittyvä	Liittyvä	LibreOffice	1